Proceedings of the 3rd Belgium Netherlands Workshop on Software Evolution (BENEVOL), Eindhoven, The Netherlands, May 26-27, 2005

Table of Contents

Preface

BENEVOL is a series of workshops for researches from Belgium and the Netherlands in the domain of software evolution. The BENEVOL workshop is a platform for researchers to present finished and ongoing research and to discuss with their colleagues. The goal of BENEVOL is to stimulate collaboration between the workshop participants. Previous editions of BENEVOL were held at the Centrum voor Wiskunde en Informatica (CWI) in Amsterdam (2003) and at the Universiteit Antwerpen (2004).

The third edition of BENEVOL was held at the Technische Universiteit Eindhoven (TU/e) on May 26th and May 27th, 2005. Researches from various institutions in Belgium and the Netherlands participated in the workshop. The workshop was organized by Christian Lange and Michel Chaudron from the System Architecture and Networking group (SAN) at the TU/e and Tom Tourwé from the CWI.

The program of the workshop contained 14 presentations that were grouped into sessions covering the following topics:

- Aspect-Oriented Software Evolution, including evolution of aspect programs, identification of aspects, and extraction of aspects. (Session chair: Tom Tourwé)
- Model-Driven Software Evolution, including model extraction, code generation, and co-evolution. (Session chair: Michel Chaudron)
- Formal Foundations of Software Evolution, including formal refactorings, and models for evolution. (Session chair: Tom Verhoeff)
- Understanding Evolution, including quality metrics, visualizing evolution, and studying change histories. (Session chair: Kim Mens)
- Tool demonstrations.

This report contains the slides of twelve of the presentations. Most presentations were revised after the workshop such that they cover the feedback from the discussions at BENEVOL. Contact information of the authors is provided in the presentations and additional information is available on the authors' websites.

We would like to thank all participants of the third edition of BENEVOL for their attendance, presentations and discussions, and Cecile Brouwer and Richard Verhoeven for their help during the organization, which made it a successful workshop. We would also like to thank the System Architecture and Networking group of the TU/e and the FWO WOG scientific research network on Foundations of Software Evolution for sponsoring the workshop.

Eindhoven, January 2006

Christian F. J. Lange
Michel R.V. Chaudron
Tom Tourwé

Participants

Bram Adams,	Universiteit Gent
Michel Chaudron,	Technische Universiteit Eindhoven
Serge Demeyer,	Universiteit Antwerpen
Peter Ebraert,	Vrije Universiteit Brussel
Andy Kellens,	Vrije Universiteit Brussel
Marc van Kempen,	Technische Universiteit Eindhoven
Christian Lange,	Technische Universiteit Eindhoven
Kim Mens,	Université catholique de Louvain
Tom Mens,	Université de Mons-Hainaut
Dennis van Opzeeland,	Technische Universiteit Eindhoven
Reinier Post,	Technische Universiteit Eindhoven
Coen De Roover,	Vrije Universiteit Brussel
Filip Van Rysselberghe,	Universiteit Antwerpen
Hans Schippers,	Universiteit Antwerpen
Tom Tourwe,	Centrum voor Wiskunde en Informatica
Tom Verhoeff,	Technische Universiteit Eindhoven
Lucian Voinea,	Technische Universiteit Eindhoven
Martijn Wijns,	Technische Universiteit Eindhoven
Roel Wuyts,	Université libre de Bruxelles

Presentations

A Qualitative Comparison of Three Aspect Mining Techniques

M. Ceccato, M. Marin, K. Mens,
L. Moonen, T. Tourwé, P. Tonella

Aspect mining

- Identification of crosscutting concerns in existing software systems
- Starting point for system exploration
- Support program comprehension, software maintenance and evolution
  - e.g. migrating to an AOP solution

Goal: high degree of automation

Comparison and combination of mining techniques

- Understand what “assumptions” (about crosscutting concerns) the techniques rely on
- Evaluate strengths and weaknesses
- Mutual filtering / completion
- Enhance automation through a multi-technique approach and tool

Three aspect mining techniques

– Identifier Analysis –

Use FCA to group classes/methods with similar names

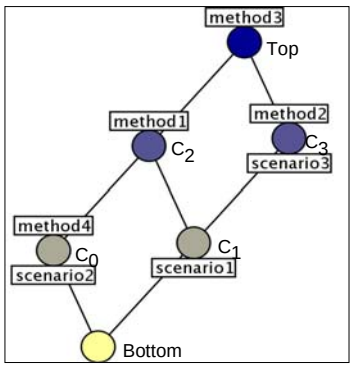
	figure	drawing	request	remove	update	change	event	...
drawingRequestUpdate(DrawingChangeEvent e)	-	X	X	-	X	-	-	...
figureRequestRemove(FigureChangeEvent e)	X	-	X	X	-	-	-	...
figureRequestUpdate(FigureChangeEvent e)	X	-	X	-	X	-	-	...
figureRequestRemove(FigureChangeEvent e)	X	-	X	X	-	-	-	...
figureRequestUpdate(FigureChangeEvent e)	X	-	X	-	X	-	-	...
...	...	...	X	...	...	...	...	...

Three aspect mining techniques
– Dynamic Analysis –

Use FCA to associate methods with the most specific use case scenarios in which they are executed

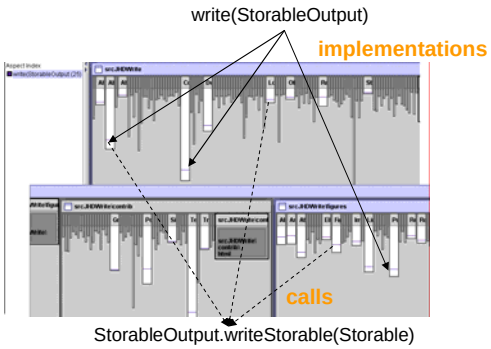
	$meth_1$	$meth_2$	$meth_3$	$meth_4$
$scen_1$	x	x	x	
$scen_2$	x		x	x
$scen_3$		x	x	

FCA



Concept lattice with sparse labeling

Three aspect mining techniques
– Fan-in analysis –

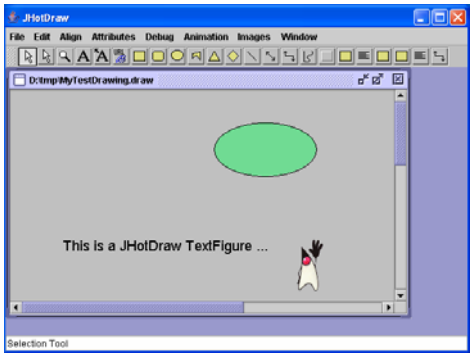


Persistence concern

Concerns

- Contract enforcement
- Consistent behavior
- Scattered implementation relying on common functionality
- Design patterns with specific structure

JHotDraw
– common benchmark for aspect mining –



- Framework for 2D graphics
- ~18,000NCLOC
- Open-source (jhotdraw.org)
- Good design – GoF patterns (Gamma et al.)

Comparison

Concern	Fan-in analysis	Identifier Analysis	Dynamic Analysis
Observer	+	+	+
Consistent Behavior / Contract Enforcement	+	-	-
Command Execution	+	+	-
Bring to front / Send to back	-	-	+
Manage Handles	-	+	+
Move Figures	+ (discarded)	+	+

Conclusions drawn from the results

- Limitations
  - **Dynamic** analysis: misses functionalities exercised by *all* traces
  - **Fan-in**: only crosscutting with large extent
  - **Identifier** analysis: relies on naming conventions
- Combination (orthogonal properties) – enhance automation and improve individual results

Combination of techniques

- **Increased coverage**
  - the union of discovered results (fan-in + dynamic)
- **Improved completeness** for the discovered aspect "seeds"
  - more elements relevant to the aspect (+ identifier)
- **Coarse-grained aspects**
  - grouping of identifier analysis concepts (fan-in/dynamic)
- **Filtering**
  - Discard irrelevant concepts

Resources

- Detailed results
  - **Fan-in**: swerl.tudelft.nl/amr
  - **Dynamic analysis**: star.itc.it/dynamo/jhotdraw-detailed-results.html
  - **Identifier analysis**: ask me ☺
- *JHotDraw* as benchmark and *AJHotDraw* as showcase for aspect refactoring
- Tools: *Dynamo*, *FanInTool*, *DelfSTof*
- Collaborations
  - *AIRCo/AIRPort*

Metadata and aspect evolution

Experiences in Aspicere

Bram ADAMS
Software Engineering Lab, INTEC, UGent

Aspicere

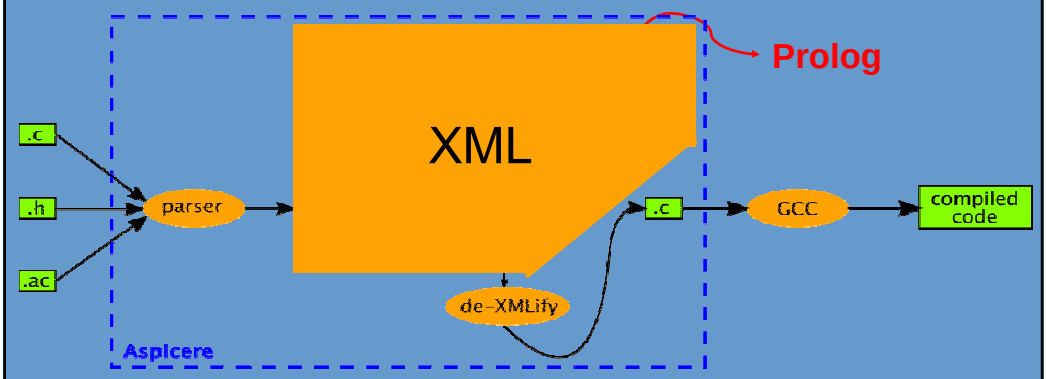
- What's in a name?
  - aspicere ≡ "to look at" (Latin)
  - Here: aspect language for C
- Characteristics:
  - Prolog-based pointcut language
  - Source code weaver
  - Currently only statically determinable joinpoints
  - Likewise no weaving within advices
- Future:
  - Merging into GCC 4.0 ("heterogeneous AOP")
  - cflow, sequence, ...
  - Weaving inside advices

2

Outline

1. Aspicere, a short introduction

General architecture



- Weaver ≡ Source-to-source transformer
  ≡ preprocessor for GCC

More details

1. Parser:
   - btyacc (backtracking): slowwwwwww ...
   - Antlr: very fast + type-checking
2. Extraction:
   - XSLT + XPath (cached)
3. Joinpoint matching (Prolog):
   - Backward chaining
   - Instantiate joinpoints as needed
   - **Bind** weave-time properties

   } WHY?
4. Weaving:
   - Depends on joinpoint type
   - Highly procedural
5. De-XMLify:
   - XML to source code

Even more details ...

source code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(pir,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```

```
void log(thisJoinPoint* jp){
    ...
}
```

```
void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

Example

BINDING

```
ReturnType advice tracing_nonvoid(ReturnType) on (Jp):
  call(Jp,_)
  && type(Jp,ReturnType)
  && !str_matches("void",ReturnType)
    {
    ReturnType i;
    /* Tracing code */
    i = proceed ();
    /* Tracing code */
    return i;
    }
```

Prolog predicates

"Templatized" C

➡ Aspect ≡ normal compilation unit enhanced with advice

Bindings

- What?
  - Logic variables which are bound and can be used freely throughout advice code
  - ≈ C++ template parameter
  - cf. Kris Gybels' and Johan Brichau's work, Cobble, LogicAJ, ...
- How?
  - Consider tuple of bindings $L=(L_1,\ldots,L_n)$
  - Instantiate advice once for all solutions for L
- Why?
  - Leverage power of Prolog ➔ reusable, robust pointcuts
  - NECESSITY ➔ no Object-class, nor template parameters

➡ **generic aspect language**

Outline

2. Metadata

Metadata

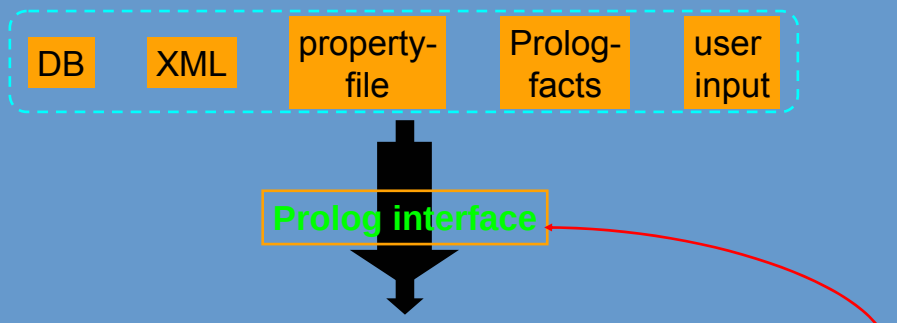
- What?
  - "data about data": semantics, design decisions, conventions, ...
- Why?
  - automated (aspectized) evolution, aspect mining, ...
- How?
  - Documentation ➔ Javadoc, Doxygen, ...
  - Separate file ➔ property files, ...
  - Language support ➔ Java 5 annotations, C# custom attributes
  - AOP introduction ➔ AspectJ 5
- In Aspicere:
  - Prolog facts & rules ≡ ... ∩ ...
- Future:
  - What about annotations in C?

🗐 loose coupling

🗐 no fixed metadata source

⌛ delocalized

Metadata supply and consumption




```
ReturnType advice serialize(ReturnType) on (Jp):
  call(Jp,Name)
  && type(Jp,ReturnType)
  && transaction(Name)
   {/*...*/ }
```

metadata

Outline

3. Demonstration

Conclusion and questions

- Conclusion:
  - Prolog facts and rules enable transparent storing of metadata
  - Aspicere's use of Prolog-like pointcuts allows easy exploitation of metadata
- Questions:
  - Does direct language support for metadata (a.k.a. annotations) yield better evolution opportunities than other mechanisms?
  - What about availability of metadata?

- Brichau, J., Mens, K. and De Volder, K. (2002). *Building composable aspect-specific languages with logic metaprogramming.* In GPCE '02: The ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering, pages 110–127. Springer-Verlag.
- Gybels, K. and Brichau, J. (2003). *Arranging language features for more robust pattern-based crosscuts.* In AOSD '03: Proceedings of the 2nd international conference on Aspect-Oriented Software Development, pages 60–69. ACM Press.
- Kniesel, G., Rho, T. and Hanenberg, S. (2004). *Evolvable Pattern Implementations Need Generic Aspects.* In ECOOP '04: Proceedings of Workshop on Reflection, AOP and Meta-Data for Software Evolution.
- Laddad, R. (2005): "AOP and metadata: A perfect match, Part 1 and 2" (IBM developerWorks)
- Lämmel, R. and De Schutter, K. (2005). *What does aspect-oriented programming mean to Cobol?* In AOSD '05: Proceedings of the 4th international conference on Aspect-Oriented Software Development, pages 99–110 . ACM Press.
- Loughran, N. and Rashid (2003). *Supporting Evolution in Software using Frame Technology and Aspect-Orientation*. Workshop on Software Variability Management, Groningen, The Netherlands.

14

Effects of Defects in UML Models

Christian Lange
C.F.J.Lange@TUE.nl
Michel Chaudron
M.R.V.Chaudron@TUE.nl

www.win.tue.nl/~clange
www.win.tue.nl/empanada

BENEVOL workshop, May 26th 2005

Overview

- Introduction and Motivation
- Experimental Design
- Results
- Analysis
- Conclusion

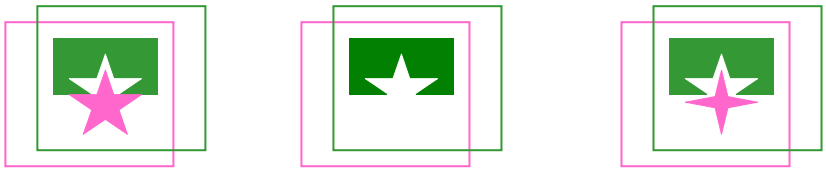
Introduction: Empanada

- EmpAnADa project
  - Empirical Analysis of Architecture and Design Quality
    - Early evaluation of quality attributes
      - Metrics for Architecture and Design models
    - Improvement by Refactoring
    - **Completeness and Consistency** checking
  - Development of methods
  - Method validation by Empirical Studies
    - Direct feedback
    - Exploring problems in practice

Introduction: UML issues

- Unified Modeling Language
  - No formal semantics
  - Offers large degree of freedom
    - Extension Mechanisms
    - 9 Diagram types (UML 2 has even 13!!)
      - Diagrams are overlapping!
  - UML is used in many different ways
  - UML Defects:
    - Completeness
    - Consistency

Introduction: Defects

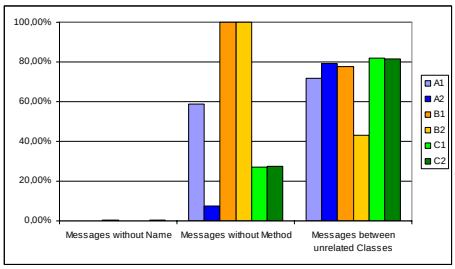


- Correct
  - Matching elements in overlapping parts of diagrams
- Completeness defect
  - Missing element
- Consistency defect
  - Mismatch
  - Conflicting elements

- Detection by checking rules
  - SAAT
  - RPA

Case Study Results

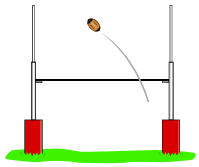
- Our case studies have shown large numbers of defects
  - How serious are defects?



Consistency	A1	A2	B1	B2	C1	C2
Messages without Name	0,00%	0,00%	0,28%	0,00%	0,00%	0,46%
Messages without Method	58,73%	7,62%	100,00%	100,00%	27,14%	27,40%
Messages between unrelated Classes	71,94%	79,37%	77,73%	43,14%	81,90%	81,74%

Purpose of Study

- Defects occur in practice, but does it matter?
  - Are defects detected in practice?
  - If not, do defects lead to misunderstandings?



- Goal (GQM)
  - *Analyze* defects in UML models
    for the purpose of identifying issues and observations
    with respect to misunderstandings and ambiguities
    from the perspective of the researcher
    in the context of TUE grad students and professionals.

Experiment Design

- Give UML models to subjects
- Ask question about UML model
  - Two types
    - Which implementation matches the diagrams?
    - How do you interpret these diagrams?
  - Answer from the perspective of the developer.
- Multiple-choice test
  - 4 options
  - + 1 option (“There’s something wrong, can’t give an answer”)
- Background questions
- Pilot run was performed

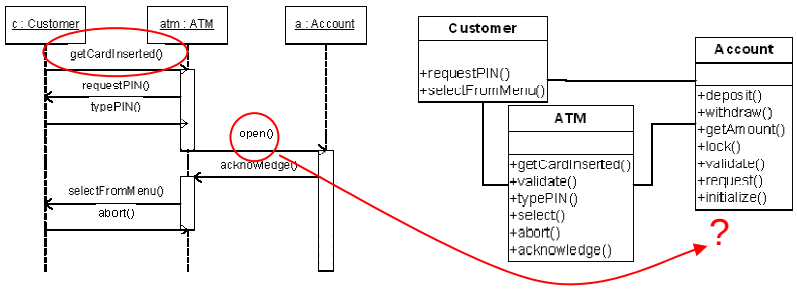
Experimental Treatment

- Treatment
  - one of 9 defects
  - no defect
- Defects are injected in model
  - Each *infected* question is paired with a *control question*
- All subjects receive all treatment levels
  - "same subject design"
  - by definition "balanced"
    - → all treatment groups have the same background

Treatments: Defects

- Message without Name
- Message Name does not correspond to Method
- Message in the wrong direction
- Class from CD not in SD
- Class from SD not in CD
- Use Case without SD
- Multiple definitions of Class with equal Name
- Method from SD not in CD
- Method from CD not in SD

Example Question

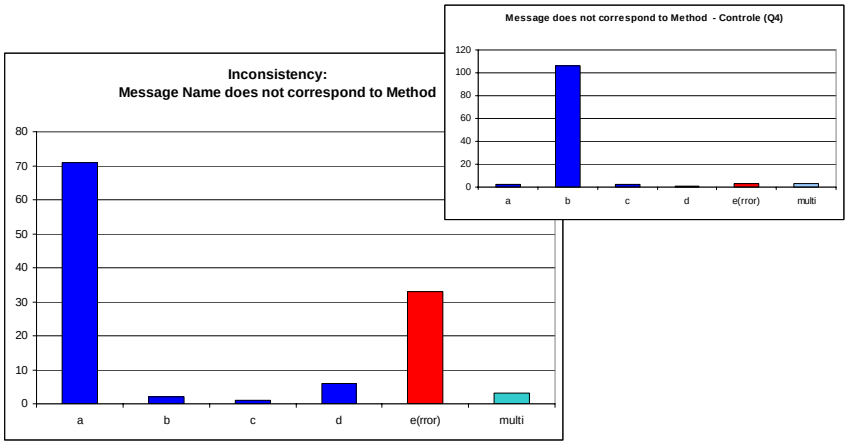


- Suppose you are developer in this banking software project. It is your task to implement class ATM. Please indicate how you would implement the ATM class given these two UML diagrams?

A) SD leading	B) CD leading	C) wrong	D) CD leading	

Class ATM{
Method
getCardInserted(){
c.requestPIN();
dosomething;
a.open()
}
…}
	Class ATM{
	Method
	getCardInserted(){
	c.requestPIN();
	dosomething;
	a.lock()
	}
	…}
		Class ATM{
		method
		getCardInserted(){
		c.requestPIN();
		dosomething;
		a.acknowledge()
		}
		…}
			Class ATM{
			method
			getCardInserted(){
			c.requestPIN();
			dosomething;
			a.validate()
			}
			…}
				Something is wrong!

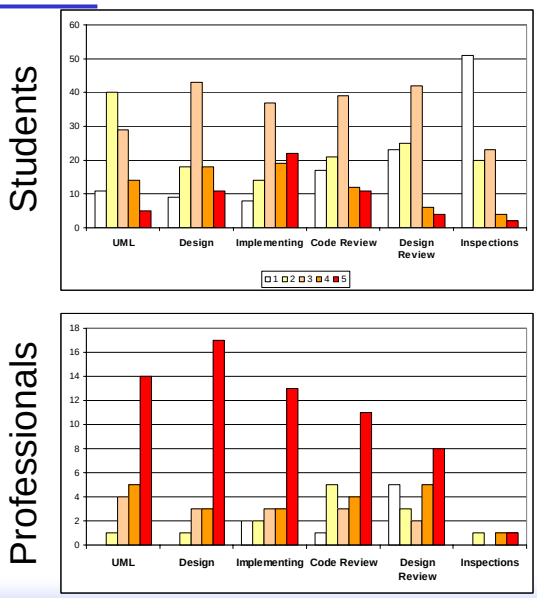
Results of Question



Subjects

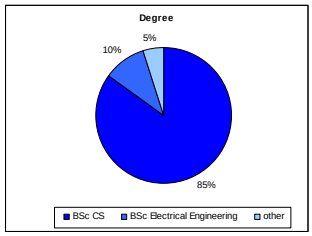
- 110 Students
  - 1st year MSc programme (TUE)
  - Course: Software Architecture (2II10)
  - Preparation
    - Lecture about UML
    - 5 weeks assignment: designing and evaluating UML
- Professionals
  - Completed online questionnaire
  - Emailed URL to contacts and newsgroups
  - 45 answered Q1 - 24 answered Q10

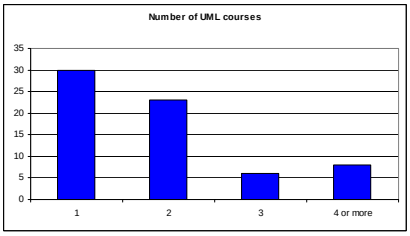
Subjects' Experience



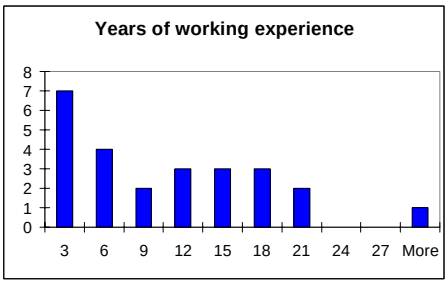
Subjects' Experience II

Students

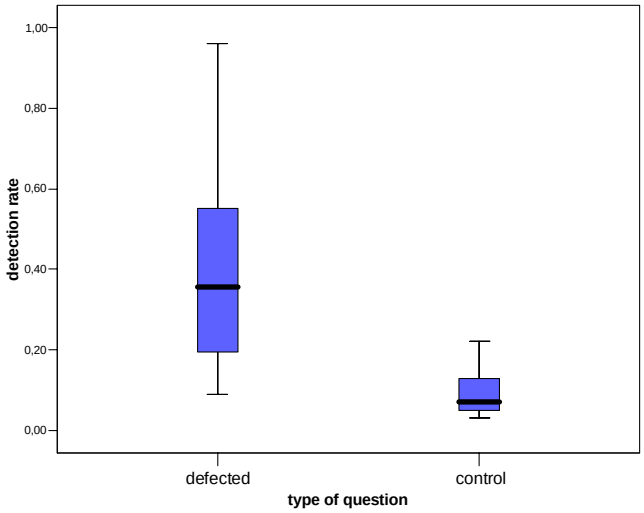




Professionals



Results: Detection



Results: Detection II

- Defects
  - Sorted by detection rate (students)

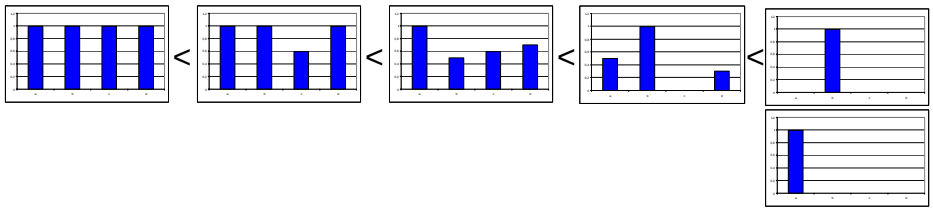
Defect	Stud.	Prof.
Class not in SD (symb.)	0,95	0,96
Message without Name	0,69	0,60
Message in the wrong Direction	0,60	0,58
UC without SD	0,50	0,52
Method not in CD	0,49	na
Message without Method (symb.)	0,49	0,33
Class not in SD	0,47	0,68
Message without Method	0,39	0,38
Class not in CD	0,18	0,11
Method not in SD	0,14	na
Multiple Class defs.	0,10	0,33
Average	*0,46*	*0,50*
Std Dev	*0,26*	*0,25*
Max	*0,95*	*0,96*
Min	*0,10*	*0,11*

Variability Measure (Entropy)

- If a defect is not detected, does it lead to misinterpretation?

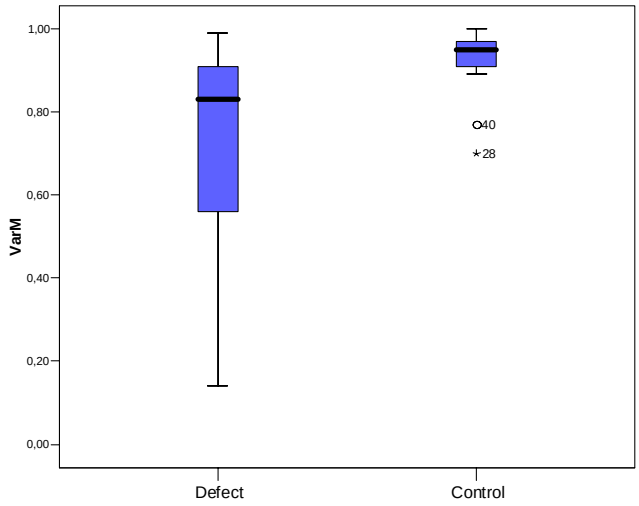
$$VarM(k_0..k_{i-1}) = 1 - 2\frac{\sum_{i=0}^{K} k_i i}{N(K-1)}$$

  - K = Number of options, N = Sum of all answers,
  - k_i = # of answers of option i (k_i 's are ordered: $k_i \leq k_{i+1}$)
- Intuition: Measuring how discriminating a distribution is



VarM=0 VarM=1

Results: Variability



Results: Variability

- Defects
  - Sorted according to VarM

Defect	Stud.	Prof.
Multiple Class defs. (meth.)	0,92	0,68
Message without Method (symb.)	0,86	0,94
Message without Method	0,84	0,90
UC without SD	0,83	0,44
Class not in CD (meth.)	0,83	0,93
Method not in CD	0,69	n/a
Method not in SD	0,67	n/a
Class not in SD	0,49	0,64
Message in the wrong Direction	0,47	0,95
Message without Name	0,47	0,44
Class not in SD (symb.)	0,34	0,14
Average	*0,71*	*0,80*
Std Dev	*0,16*	*0,21*
Max	*0,86*	*0,95*
Min	*0,47*	*0,44*

Severity

- Product of
  - Detection rate
  - VarM
- Combination of low detection rate and many different interpretations (low VarM) →causes most misunderstandings

Defect	Stud.	Prof.
Message without Name	0,37	0,26
Message in the wrong Direction	0,32	0,55
Class not in SD (symb.)	0,32	0,14
Class not in SD	0,24	0,44
Method not in CD	0,15	n/a
UC without SD	0,09	0,23
Message without Method (symb.)	0,07	0,31
Message without Method	0,06	0,34
Method not in SD	0,05	n/a
Class not in CD	0,03	0,21
Multiple Class defs.	0,01	0,23

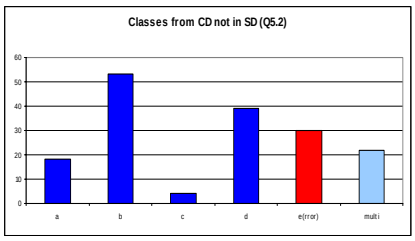
Domain Knowledge

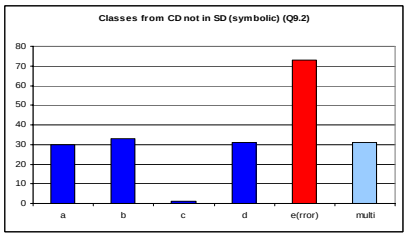
"There are no defcts in UML models that IPA members create!"

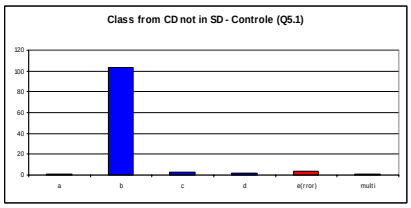
- Can defects be corrected by using context knowledge (domain knowledge) ?
- We made equal questions
  - One version with "context" (Traincrossing, Sensor)
  - One version without (Class A, Method 3)
- (Cultural background was taken into account)

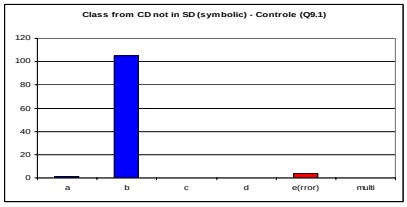
Domain Knowledge II

	Domain Knowledge	No Domain Knowledge
Detection Rate	0.47	0.95
VarM	0.49	0.34









Generalizability

- Is it valid to generalize the outcome of the students experiment?
- Pearson Correlation
  - **Between student results and professional results**

  - **For detection rate: 0.929 (p-value < 0,001)**
  - **For VarM: 0.643 (p-value < 0,004)**

  - **"significant" = p-value < 0,05**

Order of Diagrams

- In most cases the sequence diagram was regarded as the *leading* diagram
- To investigate order effects, we presented some questions in a second run with the diagrams in reversed order

- Pearson Correlation
  - Professionals vs. reversed
    - Detection rate: 0.824 (p-value: 0.044)
    - VarM: 0.899 (p-value: 0.015)
  - Students vs. reversed
    - Detection rate: 0.913 (p-value: 0.011)
    - VarM: 0.638 (p-value: 0.173)

Conclusions

- It makes sense to detect defects!
- Defects in UML models
  - are detected only to a rather low degree
  - do cause misunderstandings
- We ordered defects according to severity
- Domain knowledge matters!
  - But can cause misunderstandings
- Order of models does not matter

- Fault-injection in UML models

Lessons learned

- Experimenting is not as simple as it seems
  - Design
    - What is the question? Hypothesis?
    - What are the variables?
    - How to distribute the treatments over the subjects?
  - Preparation
    - Experiment Material
  - Carefully instructing the subjects
  - Analysis
    - Using the proper methods

?

TU/e eindhoven university of technology

Assessing Correspondence between Design and Implementation

Dennis van Opzeeland

d.j.a.v.opzeeland@student.tue.nl

Eindhoven University of Technology,
The Netherlands

/ faculty of Computer Science

TU/e eindhoven university of technology

Outline

- Introduction
- What is correspondence?
- Matching of implementation pieces to design elements
- Highlighting differences
- Case study
- Conclusion

/ faculty of Computer Science 2

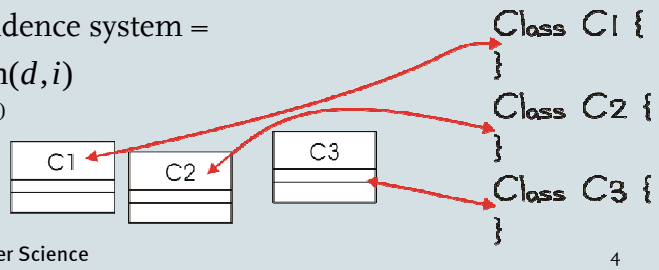
Introduction

- Correspondence:
  - Similarity between design and implementation
- Correspondence vs. evolution
  - Correspondence degrades if implementation evolves but design doesn't
  - Correspondence ↓
    - ⇒ Maintainability ↓
      - ⇒ Evolution effort ↑

What is correspondence?

- Expressed in terms of the model elements
  - Design: classes, interfaces, ...
  - Implementation: class declaration, interface specification,...
- Mapping between design elements and implementation elements
- Correspondence system =

$$\sum_{d \in D, i \in I | \mathrm{eq}(d,i)} \mathrm{sim}(d,i)$$



Typical deviations from design

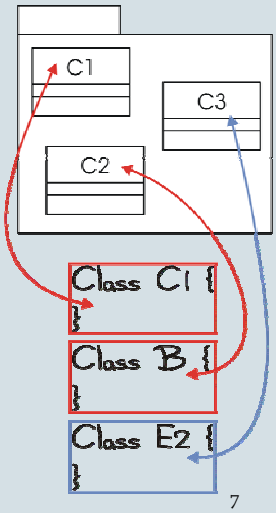
- Structural
  - Easy to check
  - Examples
    - Introduction of new classifiers
    - Differences in names
    - Introduction of new operations and attributes
    - Introduction of dependencies and associations
- Behavioral
  - Hard to check
  - Examples
    - Incompatible message sequences
- Not all deviations are equally problematic

Finding the matching

- Given:
  - Set of design classifiers
  - Set of implementation classifiers
- Problem:
  - Find the design pieces and implementation pieces that were meant to be "the same"
- Different approaches
  - Classifier names
  - Structural properties
  - Package information
  - Metric profile

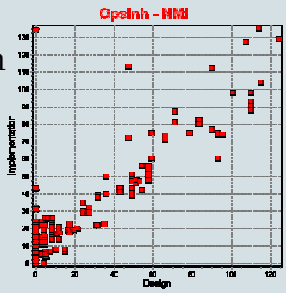
Using package information

- Heuristic:
  - Existing relations between two packages predict other relations
- Requirements
  - Development view in design
  - Directory layout for source code
  - Partial matching exists
- Purpose
  - Limit search space of other methods



Matching with Metric profiles (1)

- There exist correlations between design metrics and implementation metrics of a system
- Correlating metrics define *metric profile* of a class
  - Let c be a class, then $m(c)=(m_{1,c}, \ldots, m_{n,c})$
  - Pairwise correlations between metrics in design profile and implementation profile



Matching with Metric Profiles (2)

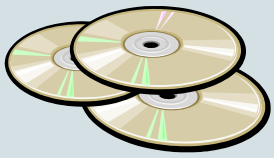
- Let d be a design class and i an implementation class
- Given metric value for design predict value for implementation metric and compare with real value

$$\mathrm{sim}(d,i) = \sum_n \rho_n \,|\, \beta_{0,n} + \mathrm{m}(d)_n \beta_{1,n} - \mathrm{m}(i)_n \,|$$

- The implementation class that fits best matches to the design class

Case study

- Characteristics
  - Industrial case
    - Firmware for DVD recorder
  - Design
    - UML 1.4
    - 346 classes
  - Implementation
    - C++
    - 777 classes
    - Lines of Code: 2,558,216
- Approach:
  - Initial matching based on names
  - Empirical analysis for metric profile approach



Correlating metrics

Design	Implementation	Corr. Coefficient
# Ops. inherited	# Ops. inherited	0.924
Depth of inh. tree	Depth of Inh. tree	0.883
Coupl. objects	Data abstr. coupl.	0.816
# Ops. inherited	# Protected ops.	0.889
# Ops. inherited	Depth of inh. tree	0.829
# Priv. operations	# Priv. operations	0.223
# Attributes	# Attributes	0.184

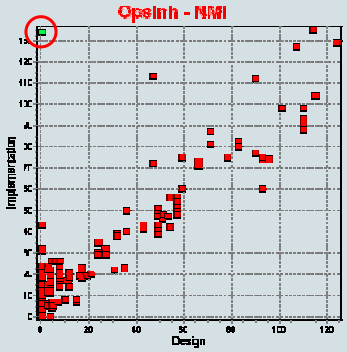
For all correlation coefficient measures, the significance level $p < 0.01$

Case study results

- Classification of deviations from design found
  - Introduction of (private/protected) attributes and operations
  - Introduction of new classes (decomposition of design classes)
  - Unused dependencies
  - Changes in inheritance tree

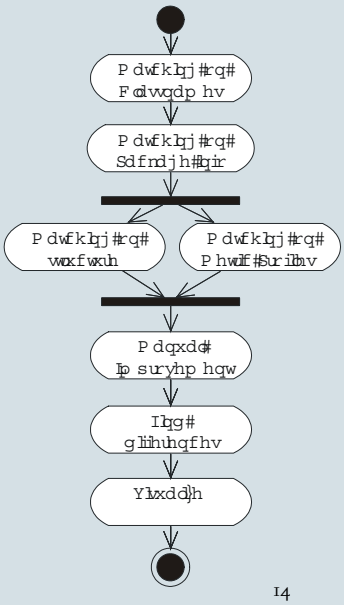
Conclusions

- Matching approaches
  - Matching based on names:
    - 77 % of design matched
    - ? % of implementation matched
  - Matching based on Metric Profiles
    - 0 % of design matched
    - 0 % of implementation matched
  - Metric Profile useful for highlighting deviations



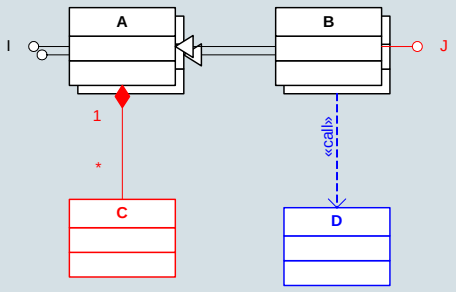
Combine strategies

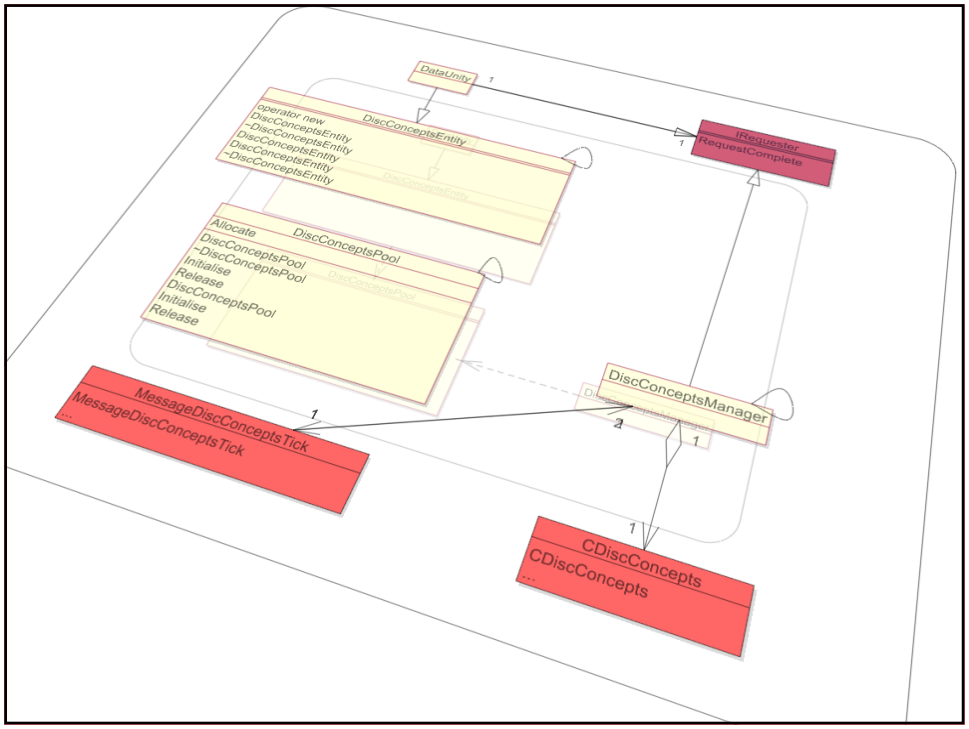
- None of the approaches defines a complete matching
- Find initial matching using a good approach
- Cluster classifiers using package information
- Apply other matching approaches on clusters
- If everything else fails: human intelligence



Visualization of differences

- Given a mapping, finding differences is quite straightforward
- Visualization using MetricView
- Overlay diagrams



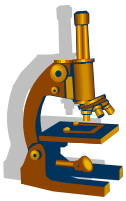


Further work

- What can be done to prevent correspondence issues?
- How can correspondence be established?
- What is the impact of correspondence issues?
- How much correspondence is needed?
- What about clustering

MetricView Evolution

- Monitoring Architectural Quality -



Martijn Wijns
M.A.M.Wijns@student.tue.nl

Within EmpAnADA project
Supervisors: Michel Chaudron & Christian Lange
Technische Universiteit Eindhoven

Outline

- Motivation & Context
- User Profiles
- Goals & Existing Tooling
- Prototyped Ideas
- Tool Demo
- Findings
- Future Work

2

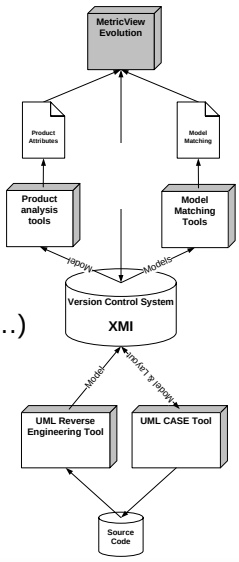
Motivation & Context

- Motivation
  - Metrics offer benefits but are hard to use
  - Metrics are often separated from model
  - Metrics offer a low abstraction level
  - Current Metric tooling focuses on single point in time

3

Motivation & Context

- Context
  - Data:
    - UML/XMI standard, many dialects
    - External Metrics
    - Matching Data
  - Tooling:
    - CASE (Rose, Together, EA, …)
    - Reverse Engineering (Columbus, …)
    - Version Control (SVN, CVS, …)
    - Correspondence (DICT, …)
    - Analysis (SAAT, SDMetrics, …)



4

User Profiles

- Three types of user profiles
  - Software Architect
    - Monitor Quality over time
    - Identify Weak Points
  - Project Manager / Client
    - Monitor Quality from a high level perspective
  - Maintainer
    - Identify weak points with their cause
    - Estimate impact of needed changes

5

Goals & Existing Tooling

- Main Goals
  - Monitor Quality
    - Support analysis for multiple versions
  - Provide Abstraction from Metrics
    - Aggregation on Multiple levels
    - Tailorable, using Custom Quality Model
  - Provide Easy access to Metrics
    - Fast and Accurate Navigation
    - Short feedback cycle
      - Keep Model and Metrics together
      - One tool to extract and visualize them

6

Goals & Existing Tooling

- Existing UML Metrics Tooling
  - SAAT
    - ☑ Easy Metric Definition: Relational Database / SQL
    - ☑ Maintainability (Dialects…): XSLT / Perl parser
    - ☒ No Visual feedback
    - ☒ Hard to install: Dependencies
  - SDMetrics
    - ☑ Fast
    - ☑ Many metric definitions / Design rules
    - ☒ Visualization rather basic
  - MetricView
    - ☑ Integrated Model and Metrics
    - ☑ Visualization
    - ☒ No integrated Metric calculation

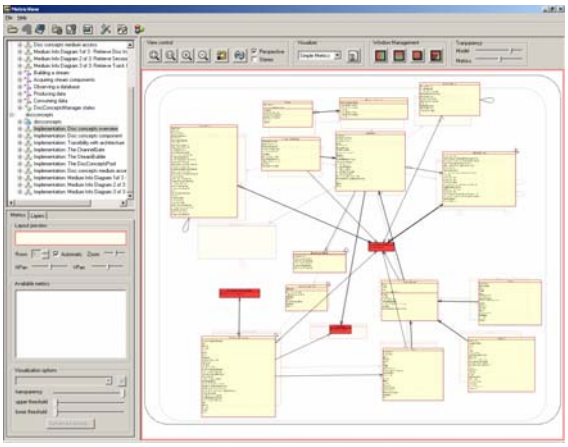
7

Prototyped Ideas

- Layout-Adjustment
- Correspondence Visualization
- Metrics over time
- Quality Tree
- Timeline
- Lange-Diagram
- Context Diagram
- Search & Highlight

8

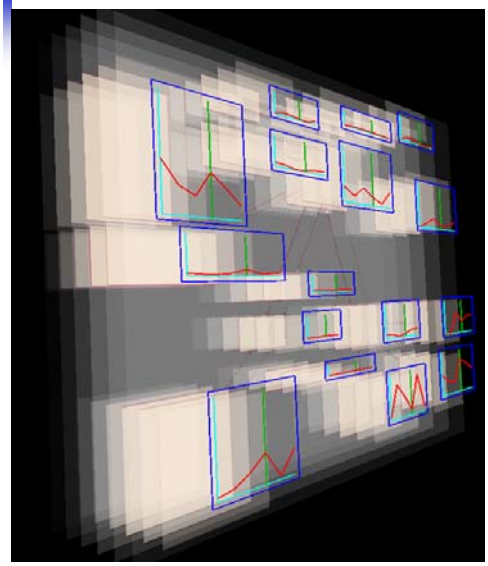
Prototype - Correspondence



- ☑ Quickly spot differences
- ☒ Layout construction

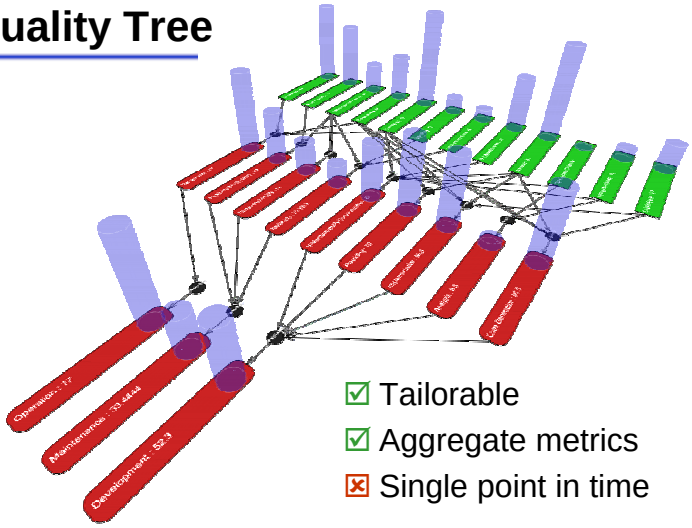
- ❑ Secondary goal: Explore MetricView implementation

Prototype – Metrics over Time



- ☑ Metrics changing over time
- ☒ Changes in structure less clear in case of many versions

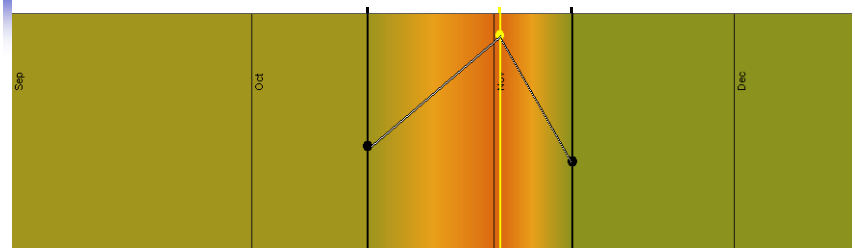
Quality Tree



- ☑ Tailorable
- ☑ Aggregate metrics
- ☒ Single point in time

11

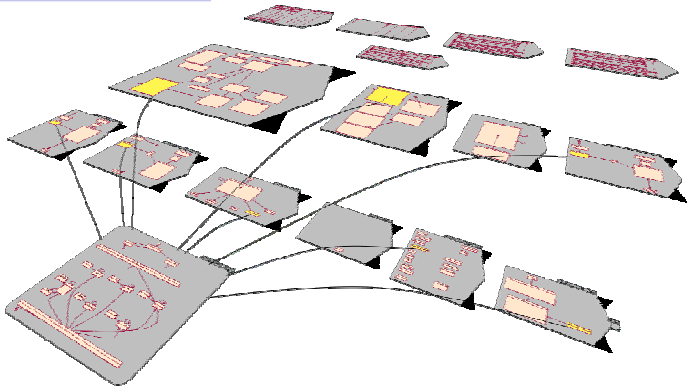
Timeline



- ☑ Visualizes the change of Metric or Quality Attribute values over Time
- ☑ Also usable for higher abstraction levels
- ☒ Does not scale well to multiple attributes

12

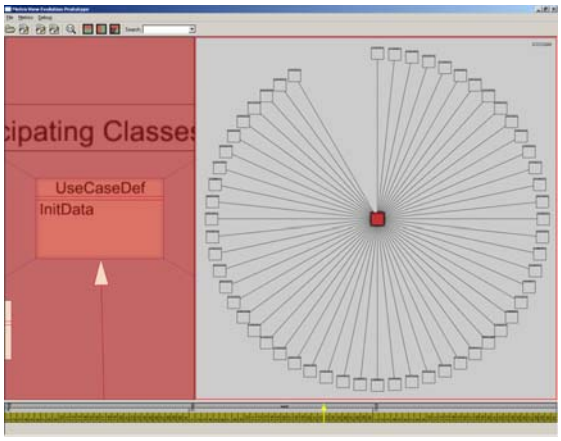
Lange-Diagram



- ☑ System Overview, including inter-diagram relations
- ☒ Layout Scalability

13

Context Diagram



- ☑ Context in complete model, not just one diagram

14

Tool Demo

DEMO

15

MetricView Evolution

- Benefits:
  - Overview of relations between diagrams
  - Context-Diagram reveals obscured problems
  - Highlighting same element in multiple diagrams
  - Search by keyword for quick navigation
- Disadvantage
  - Class level metrics give too much detail
  - Scalability for large models

16

Findings

- Suggestions for usage
  - Pattern Detection (context diagram)
  - Impact Prediction (context diagram)
  - Implicit Relations (search & highlight)
- Incomplete models
  - Design Decision not model everything

17

Future Work

- Improve Visualizations
  - Higher Abstraction levels
  - Reports
  - Dynamic Layout
  - Design Smells
- Multiple Version support
  - Specific Metrics?
  - Integrate with CVS-like tools
- Validation
  - Integration
  - During development, rather than afterwards
  - Usability Testing

18

Questions

?

19

Introduction to Software Metrics

- What are software metrics?
  - Specification for Quantification of Software Attributes
- What can you do with them?
  - Understand
  - Control
  - Improve
- Who wants them?
  - Anyone who wants to *systematically* understand, control or improve software quality
- Why Architecture Metrics?
  - Early detection means easier/cheaper to fix

20

Roel Wuyts
Université Libre de Bruxelles

Benevol, May 26th 2005 (Eindhoven, The Netherlands)

Type Reconstruction

- Context: program understanding in dynamically typed languages
  - e.g. extraction of class diagrams
- Type Reconstruction
  - input: program without types
  - output: program with types

2

Trade-offs

- Precision vs. efficiency
  - We chose efficiency for usage in a development browser
  - Use Heuristics as basis for the reconstruction
    - instead of full reconstruction

3

Heuristics

- Direct sends to instance variable
- Indirect sends to instance variables (getter methods)
- Direct assignment expressions
- Indirect assignment expressions (setter methods)
- (Type snooping)

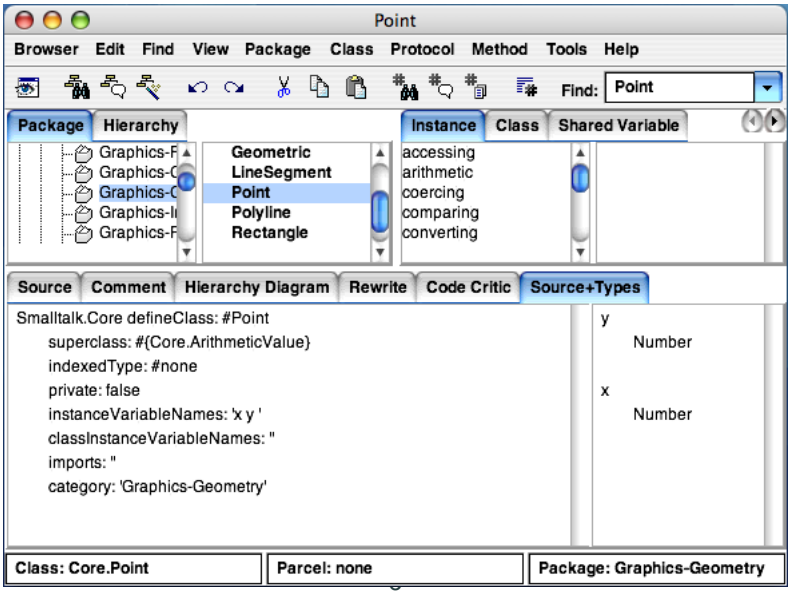
4

Implementations

- Using LiCoR (Library for Code Reasoning) in SOUL
  - on the parse tree
  - average: 500 milliseconds / instance variable
  - more elaborate and easier to extend
- Using partial evaluation on the byte code
  - average: 30 milliseconds / class

5

Demo



Conclusions & Future Work

- Works
  - *About 80%* of correctness on built-in libraries
  - Better on domain-specific code
- Future Work
  - Fix *About*
  - Will do this on (untyped) Java code and compare results

7

Challenges in Software Evolution

Tom Mens

http://w3.umh.ac.be/genlog
Software Engineering Lab
University of Mons-Hainaut
Belgium

Challenges in Software Evolution

- The presented results are the outcome of the ChaSE 2005 workshop
  - Financed by ESF and ERCIM
  - April 2005, Bern, Switzerland
- Scientific goals
  - to discuss about and identify the main challenges in software evolution
  - to address the above goal from different points of view

Classification of challenges

- Multidimensional classification
  - Time dimension
    - Short-term, medium-term, long-term research
  - Type of software evolution research
    - Managing software evolution
    - Understanding software evolution
    - Analysing software evolution
  - Interested stakeholder
    - Manager, end-user, developer, teacher, tool builder, software engineer, …
  - Type of artifact under study
    - Formalism, tool, model, language, process, people, …
  - Type of support provided
    - Same list as before…

Preserving and improving sw quality

- How can we provide tools and techniques that preserve or even improve the software quality, whatever its size, complexity, level of abstraction?

Michel Chaudron
Peter Ebraert
Kim Mens

time	research type	stakeholder	artifact	support
long	Controlling & supporting evolution	developer, project manager, end-user	Software system	tools, techniques, formalisms, processes

Supporting model evolution

Christian Lange

- Design and modeling tools provide little evolution support
- evolution techniques needed at higher level of abstraction
  - A&D models, SW architectures, requirements, …
- Model-driven engineering makes this challenge very relevant

time	research type	stakeholder	artifact	support
short	controlling, supporting	software engineer	models	tools, techniques, formalisms

Supporting co-evolution

Christian Lange, Dennis Van Opzeeland

- We urgently need better techniques to achieve co-evolution
  - Synchronisation, consistency maintenance, inconsistency management, traceability, change propagation, …
- between different types of software artifacts or different representations
  - Programs and design models
  - Software and the organisation
  - Software and languages, tools, platforms

time	research type	stakeholder	artifact	support
medium	controlling, supporting	software engineer	any pair of related artifacts	tools, techniques, formalisms

Formal support for evolution

Kim Mens,
Tom Mens

- Some formal methods not amenable for an evolutionary setting
  - e.g., no incremental verification
- Formalisms for specific evolution activities needed
  - e.g., for refactoring

time	research type	stakeholder	artifact	support
medium - long	all	researcher	formalisms	formalisms

Need for empirical research

- empirical and statistical studies needed to assess impact of process models, tools, languages, and people's experience on software evolution

Christian Lange
Dennis Van Opzeeland

time	research type	stakeholder	artifact	support
long	analysing	researcher	any evolving artifact	statistical models, empirical studies

Need for evolution benchmarks

Kim Mens

- Commonly agreed representative benchmark or case studies to compare the developed formalisms, tools, techniques on relevant and typical problems
- Getting data from industrial setting is not easy, but there are many open-source, long-lived, industrial size projects

time	research type	stakeholder	artifact	support
short - medium	understanding, comparing	researcher	software system	benchmarks, exemplars, cases

Runtime evolution

Bram Adams

- There is a need for proper support of runtime adaptations that allow systems to evolve while they are running, without needing to pause them or shut them down
  - Reflective techniques, metadata

time	research type	stakeholder	artifact	support
short - medium	Controlling, supporting	developer, end-user	languages, execution platforms	languages, execution platforms, programs

Challenge: Teaching software evolution

- How can we introduce the ideas and techniques of evolution into our educational system?
  - What do we want to teach to our students?
  - How can we teach this?
  - Where does it fit in the CS curriculum?

time	research type	stakeholder	artifact	support
short	teaching	teacher, student	any	slides, exercices, case studies, tools, books, …

A common software evolution platform

- Proposed research solutions need to scale up to long-lived industrial-size software system
- Required tools are too complex to be built in isolation
- Need for a common platform, tool integration, exchange formats, standards and so on
- Candidates: MOOSE, Eclipse

time	research type	stakeholder	artifact	support
medium	Applied research	researcher	programs	tools, frameworks, platforms, standards, exchange formats

Evolution as a language construct

- Change should be a first-class entity in programming or modelling languages
- Evolution support is easier in dynamically typed languages with reflective capabilities

time	research type	stakeholder	artifact	support
short - medium	controlling, supporting	language designer, tool builder, researcher	languages	languages and programs

Supporting multi-language systems

- Many complex and large systems are built using 3 or more languages
- Evolution techniques should be parameterisable on or independent of the language

time	research type	stakeholder	artifact	support
medium - long	controlling, supporting	tool builder	languages, software systems	tools, standards

Integrating change into dev. process

- Change must be integrated into conventional development process models
- Some, like agile development, already embrace change as essential
- Others, like the staged life-cycle model, have explicit support for evolution

time	research type	stakeholder	artifact	support
medium	managing, controlling	manager, software engineer	software process model	software process model

Increasing manager's awareness

- Managers have to realise the importance and inevitability of software evolution
- Teach them how to plan, organise and control projects to cope with change

time	research type	stakeholder	artifact	support
short	motivating	manager, researcher		metaphors

Developing better versioning systems

- Many analyses of software evolution based on CVS or related tools
  - these weren't built for that purpose and don't store enough information
- New techniques and tools needed for recording the evolution of a system
- SCM is related

time	research type	stakeholder	artifact	support
short	analysing	tool builder	version control tools	tools

Integrating and analysing evolutionary data

- Scattered information about system changes
  - bug reports, source code, documentation, configuration files, …
- very large data sets, especially for long-lived systems
- Need efficient and heterogeneous techniques
- Extensible meta-models, data mining, and bioinformatics may be relevant

time	research type	stakeholder	artifact	support
medium	analysing	researcher	all relevant info about sw system's evolution	techniques, tools

A theory of software evolution

- Lehman has developed laws, but they need to be formalised and enriched
- May borrow ideas from evolutionary biology or linguistic evolution
- The what (noun) and how (verb) of evolution are still mostly disconnected
- How does the gathered data inform tools, techniques and formalisms?

time	research type	stakeholder	artifact	support
long	understanding, analysing	researcher	everything	theories, formalisms, laws, …

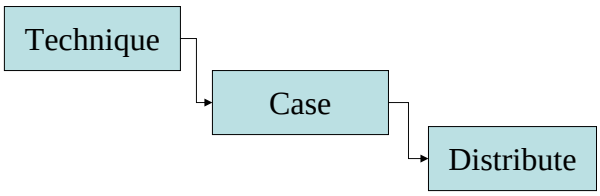
Software Evolution Case Studies

Filip Van Rysselberghe

Benevol 2005

Case study

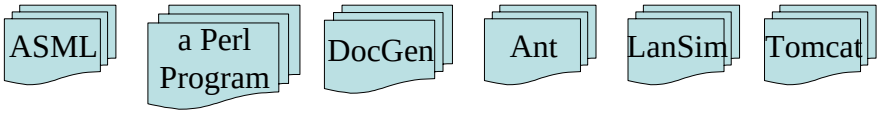
"Aspect mining research results will have to be validated by means of a series of case studies."



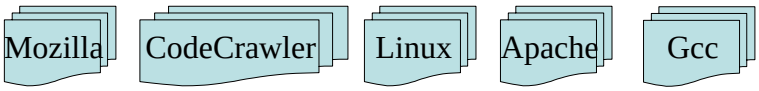
= illustrate the applicability on a concrete project

Examples

Benevol



Icsm04



Comparison

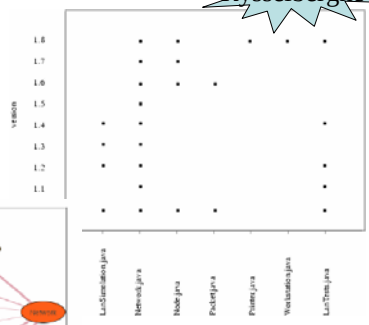
- A case per group/technique
  - We can't compare using a different basis!
  - We can't build on each other's findings
- Are we really that different?
  - Understanding, quality of architectures, aspect mining,..
  - It's all about Software Evolution!
  - ⇒ Similar solutions?
  - ⇒ Common problems?

LanSimulation

Which components are strongly coupled?

Lanza

Rysselberghe



Ratzinger

Representative

- Common evolution characteristics?
  - What's typical/common?
  - Enough information about evolution?
- Characteristics of the case?
  - How do we measure?
  - Amount of info available?
  - When?

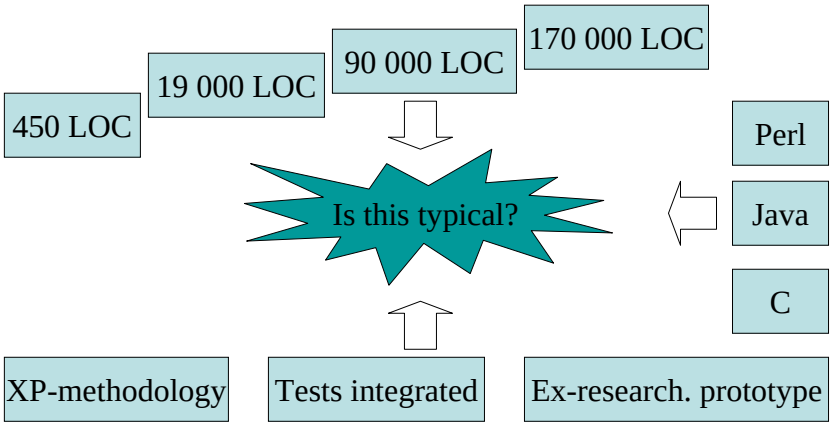
Retrospective View

Common	Case (Tomcat)
• Life-cycle • Constant change • Feature requests • Large/Coarse	

Retrospective View

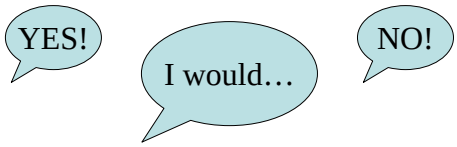
Common	Case (Tomcat)
• Life-cycle • Constant change • Feature requests • Large/Coarse	• Build from previous product • Constant change • Implements specification • Limited/Coarse • Java • Design docs/rationale • Mailing Archives

Your characteristics



Literature characteristics

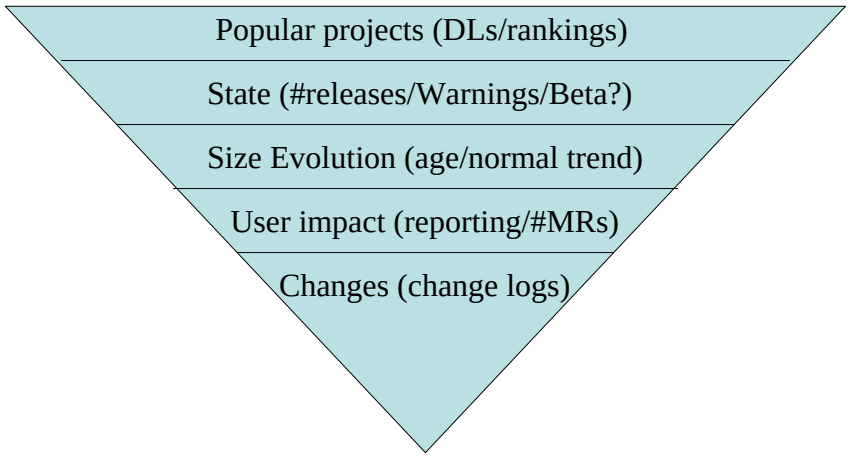
- System serves a user base
- User base has changing demands
- Generally grows
- Constant change (bugs/functional)



Case Selection

- How do we select our cases?
  - Based on measures? Which?
  - Based on quality of information?
- Where do we search?
- How did you act in the past?

Maybe like this?



Conclusion

- Many open questions!
  - Common Characteristics?
  - Case Selection?

Discuss

speak with others about it;
talk it over in detail ;
have a discussion ;

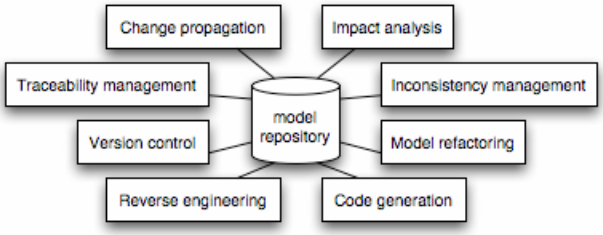
Analysing refactorings with graph transformation theory

Tom Mens

http://w3.umh.ac.be/genlog
Software Engineering Lab
University of Mons-Hainaut
Belgium

Introduction - Software Evolution

- More and better tool support needed for software evolution
  - At all levels of abstraction (e.g. programs and models)
  - For a variety of different activities



Introduction - Software Evolution

- Formalisms can be helpful for such evolution support
  - Description logics
    - For model inconsistency management
      - collaboration with R. Van Der Straeten, VUB
  - Graph transformation
    - For supporting software refactoring
    - Reasoning about preservation properties
      - collaboration with D. Janssens and S. Demeyer, UA
    - Analysing refactoring dependencies
      - collaboration with G. Taentzer and O. Runge, TU Berlin

Graph transformations

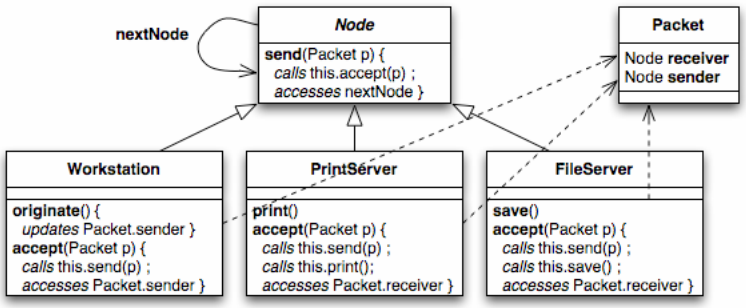
- GT theory theoretical results can help during analysis of model refactorings
  - type graph, negative application conditions, parallel and sequential (in)dependence, confluence and critical pair analysis
- GT tools allow us to perform concrete experiments
  - AGG (in collaboration with Berlin)
- Current focus
  - Analysing dependencies between class diagram refactorings

Analysing refactoring dependencies

- Concrete Scenario: Suggest refactoring opportunities
  - What are the alternatives of a selected refactoring?
  - Which other refactorings need to be applied first in order to make the selected refactoring applicable?
  - Which other refactorings are still applicable after applying the selected refactoring?
- Goal: Automate the detection of
  - mutual exclusion relationships between refactorings
  - sequential dependencies between refactorings

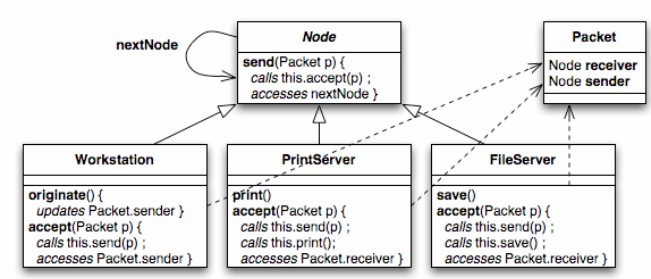
Analysing refactoring dependencies

- Example



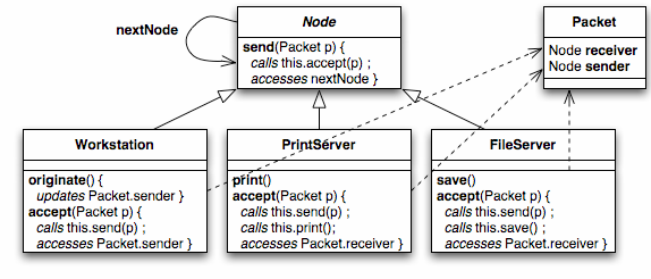
Analysing refactoring dependencies

- Refactoring opportunities
  - T1 Rename Method print in PrintServer to process
  - T2 Rename Method save in FileServer to process
  - T3 Create Superclass Server for PrintServer and FileServer
  - T4 Pull Up Method accept from PrintServer and FileServer to Server



Analysing refactoring dependencies

- Refactoring opportunities
  - T5 Move Method accept from PrintServer to Packet
  - T6 Move Method accept from FileServer to Packet
  - T7 Encapsulate Variable receiver in Packet
  - T8 Add Parameter p of type Packet to method print in PrintServer
  - T9 Add Parameter p of type Packet to method save in FileServer

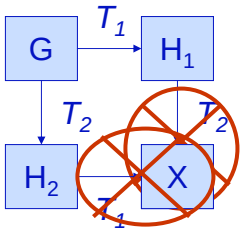


Analysing refactoring dependencies

	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	×	←		←				»	
T2		×		←					»
T3			×	←			×		
T4				×	×	×			
T5					×	×			
T6						×		×	×
T7							×	←	
T8								×	×
T9									×

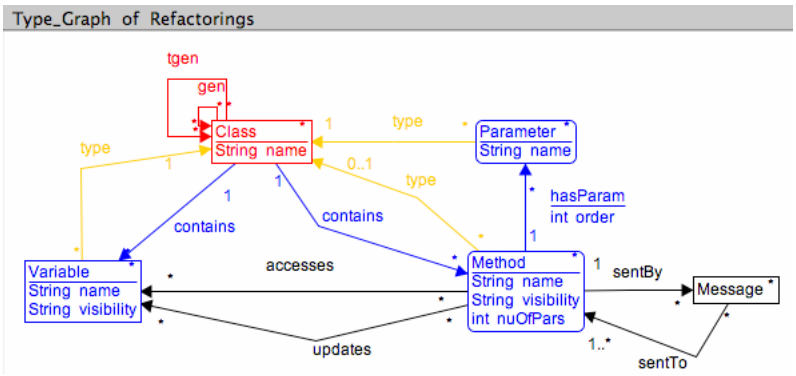
Applying graph transformation theory

- Approach: Use critical pair analysis in AGG
  - T_1 and T_2 form a *critical pair* if
    - they can both be applied to the same initial graph G but
    - applying T_1 prohibits application of T_2 and/or vice versa



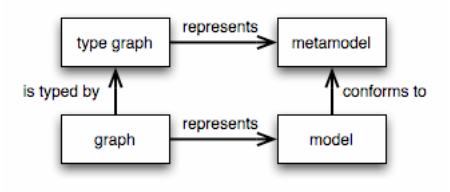
Applying graph transformation theory

Step 1: Express object-oriented metamodel as (attributed) type graph



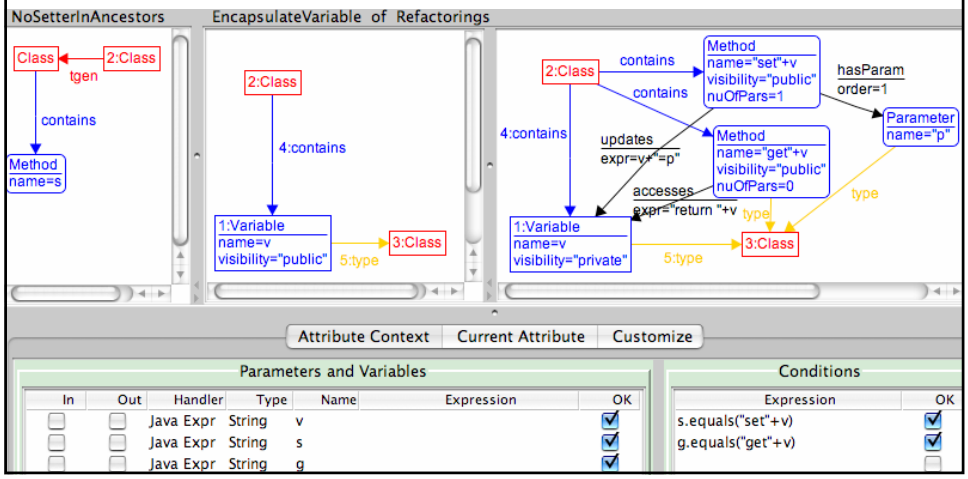
Interludium

- Type graphs versus metamodels



Applying graph transformation theory

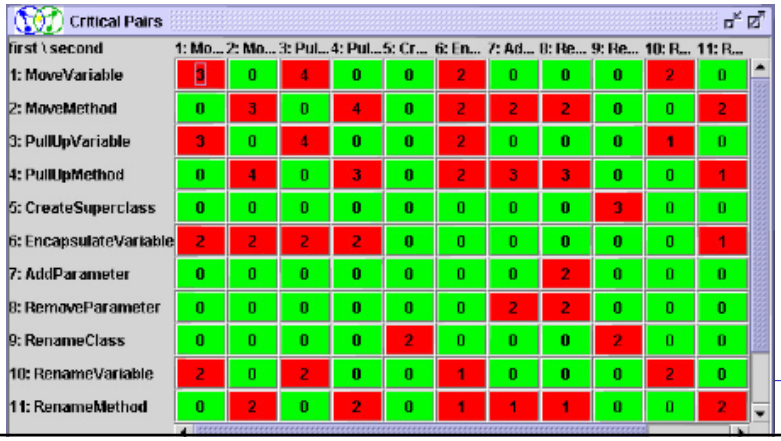
Step 2: Express refactorings as (typed attributed) graph transformations



Applying graph transformation theory

Step 3: Detect critical pairs between refactoring transformations

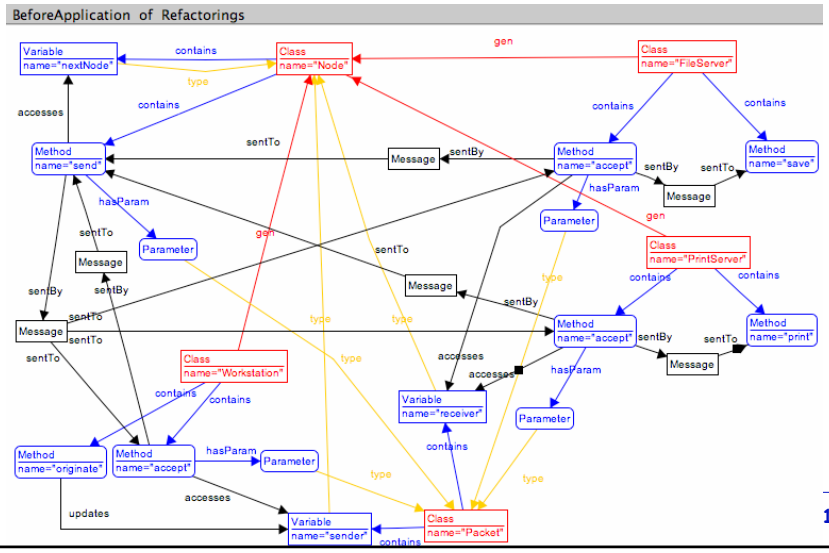
- *Potential conflicts* between refactorings



14

Applying graph transformation theory

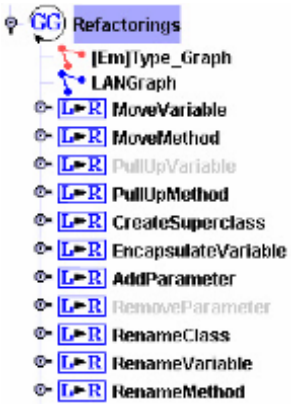
Step 4: Fine-tune critical pairs in context of concrete input graph



Applying graph transformation theory

Applying graph transformation theory

- Step 5: Perform sequential dependency analysis
  To identify dependencies between refactorings that are applicable



Conclusion

- Graph transformation theory is a suitable formalism for understanding software refactoring

Graph Transformation	Refactoring
type graph, invariants	wf-constraints
negative application conditions	preconditions
parameterised graph production with NACs and context conditions mechanism	Refactoring transformation
Critical pair analysis	Detecting mutual exclusion
Confluence analysis	Detecting sequential dependencies

technische universiteit eindhoven

Refactoring Architectural Style

using KWIC as a case-study

Marc van Kempen
Michel Chaudron

technische universiteit eindhoven

Outline

Introduction / Motivation

Case Study: KWIC

Basics of CSP

Pipe & Filter → BlackBoard

Client-Server → BlackBoard

Correspondence to implementation

Conclusions

technische universiteit eindhoven

Introduction

What is Refactoring?

What is Architectural Style?

- Codification of common recurring patterns of software design
- Construction paradigms for (a set of) design dimensions
- A vocabulary of components, connectors and constraints on these can be combined (structure).
- Examples: Pipe and Filter, Client/Server, Blackboard

technische universiteit eindhoven

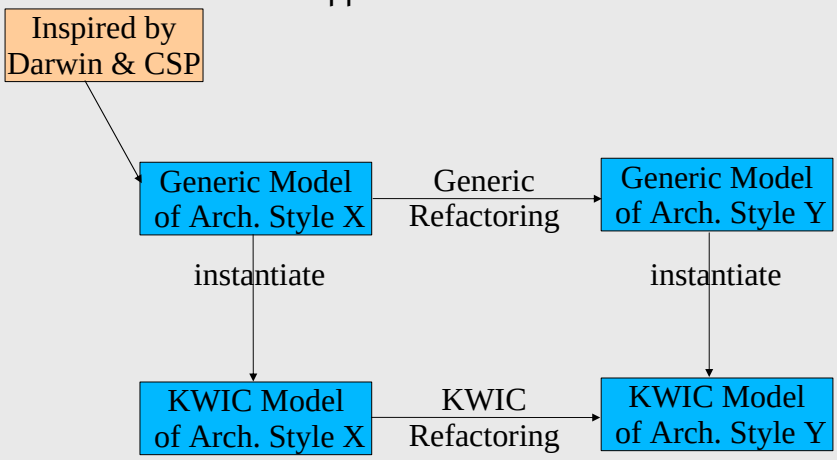
Motivation/Contributions

Motivation:

- Architecture-level refactoring
  - Useful for system migration, integration of large systems
  - Repair “mistakes” early in the development process
- Understanding architectural styles
  - Relation between styles (commonalities/differences)
- Research Separating Styles from Components
  - Exogeneous styles enhance component usability
  - Avoid architectural mismatch => ease evolution

technische universiteit eindhoven

Approach



technische universiteit eindhoven

Connecting Connectors and Components

What is a connector?

- Intermediates messages between components
- Shields component from interaction protocol

What is a component?

- Lots of definitions: “A unit of composition with clearly specified interfaces and explicit context dependencies only”
- Aim: components should be reusable; hence unchanged by refactoring

Use (adapted) Darwin notation to describe connections between connectors and components.

technische universiteit eindhoven

Keyword In Context (KWIC)

Based on a paper by Parnas (1972)

Garlan & Shaw (1994) use KWIC to illustrate the influence of Architectural Styles on Architectural Design

Good case to study refactoring between styles

technische universiteit eindhoven

KWIC (2)

What is KWIC?

Algorithm to generate alphabetically sorted permutations of an input sentence.

A permutation of a sentence is a sentence where the words are shifted one position to the right, the rightmost word is put at the beginning of the sentence. Repeat until the original sentence is reached.

KWIC(3)

Example input:

```
"Refactoring architectural styles"
```

Permutations:

```
"Refactoring architectural styles"
"Styles refactoring architectural"
"Architectural styles refactoring"
```

TU/e technische universiteit eindhoven

KWIC(4)

Sorted this becomes

```
"Architectural styles refactoring"
"Styles refactoring architectural"
"Refactoring architectural styles"
```

Which is also the expected output of the kwic process

technische universiteit eindhoven

CSP

What is CSP? Communicating Sequential Processes

- Process algebra
- Main operators:
  - Prefix: a → P
  - Parallel composition: ||
  - Communication actions: *out!v* and *in?x*
  - External choice: (a → P) [] (b → Q)

Why use CSP to describe the processes?

- Formalization
- Tools exist to prove properties of model

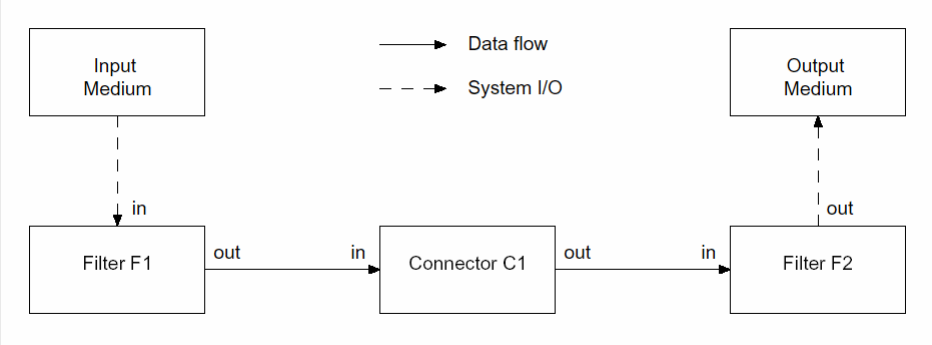
technische universiteit eindhoven

Pipe and Filter

What is pipe and filter?

technische universiteit eindhoven

Pipe and Filter architecture



technische universiteit eindhoven

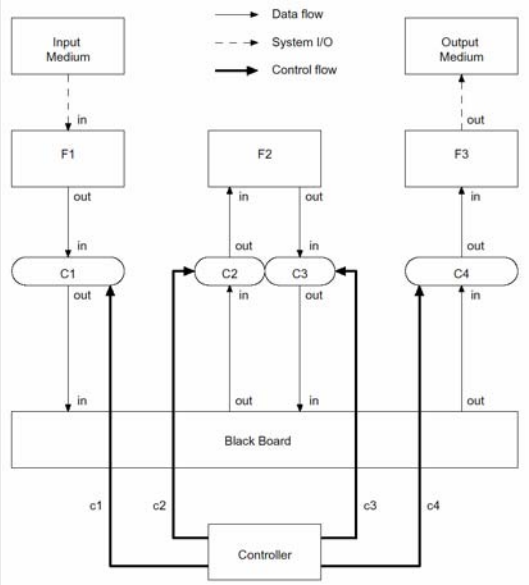
CSP definition of Pipe and Filter

Define Filter = in?x → out!f(x) → Filter
IN = out!x → IN
OUT = in?x → OUT
Connector = C(‹›)
C(‹›) = in?x → C(‹x›)
C(s) = in?x → C(s^‹x›) ‹| x ≠ eof |› C'(s)
C'(‹›) = SKIP
C'(s) = out!head(s) → C'(tail(s))

Let Filter F1, F2
Connector C1
IN.out -- F1.in
F1.out -- C1.in
etc.

Then PF = IN || F1 || C1 || F2 || OUT

Blackboard Architecture Style



technische universiteit eindhoven

CSP definition of Blackboard

IN, OUT, Filter remain the same!

Define Connector = in?x → ctrl!x → out!x → Connector

BB(‹›) = in?x → BB(‹x›)

BB(s) = (in?x → BB(s^‹x›)) [] (out!head(s) → BB(tail(s)))

Controller = c1?x → Controller ‹| x ≠ eof |› Controller1

Controller1 = c2?x → c3?x → Controller1 ‹| x ≠ eof |› Controller2

Controller2 = c4?x → Controller2 ‹| x ≠ eof |› SKIP

Let Filter F1, F2, F3

Connector C1, C2, C3, C4

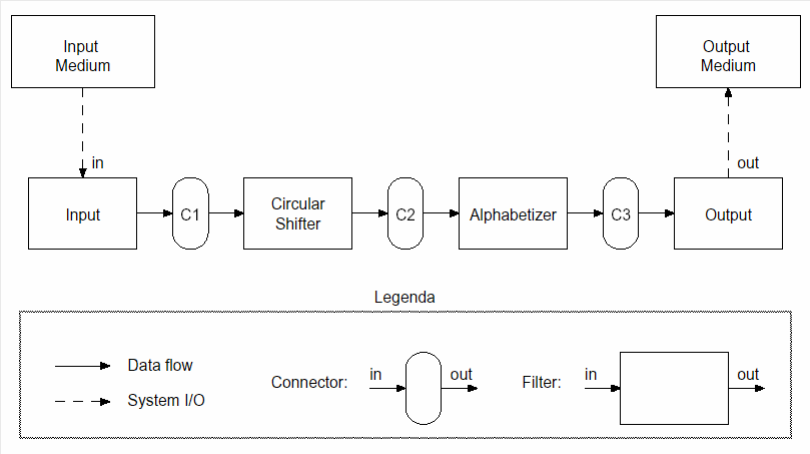
Controller Ctrl

BB BB1, BB2

Then Blackboard = IN || F1 || C1 || BB1 || C2 || F2 || C3 || BB 2 || C4 || OUT

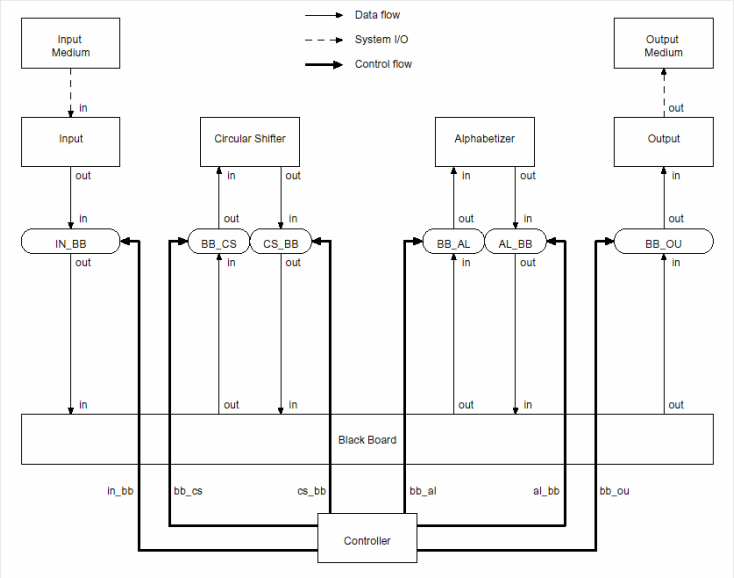
technische universiteit eindhoven

KWIC in Pipe and Filter architecture style



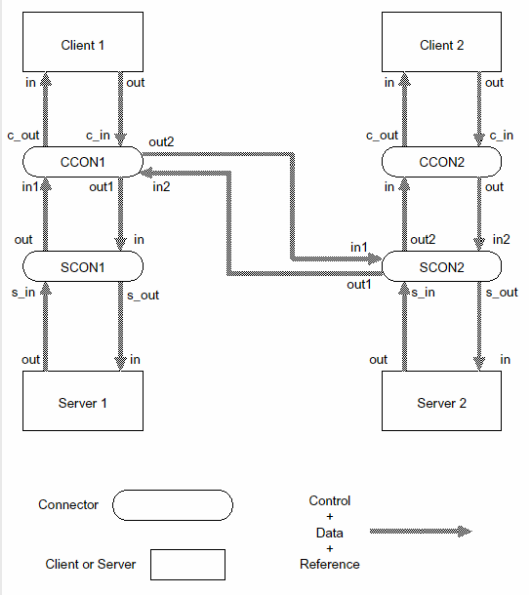
technische universiteit eindhoven

Refactor P&F to BB



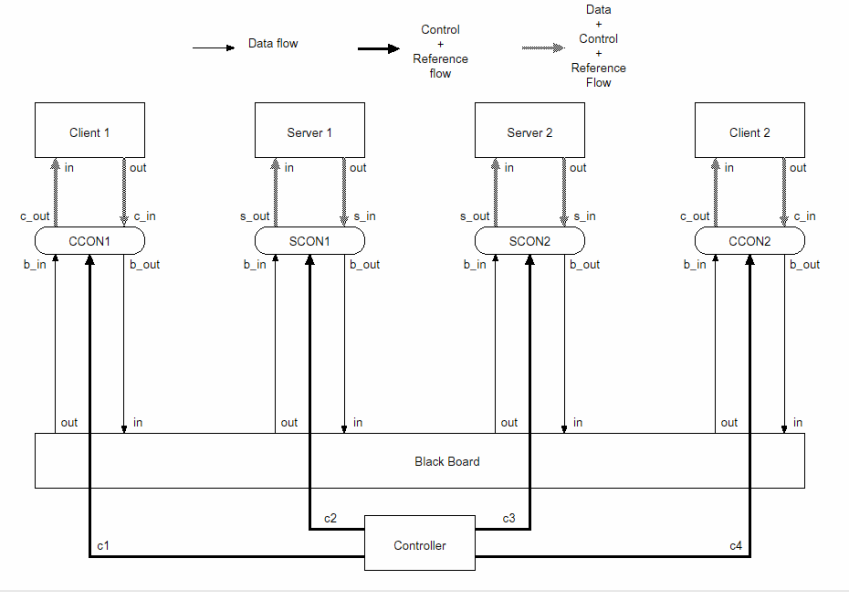
technische universiteit eindhoven

Refactor Client-Server to Blackboard



technische universiteit eindhoven

Client/Server refactored to Blackboard



In Java

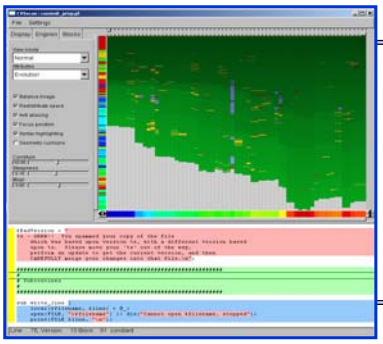
Java implementation is forthcoming, watch website.

TU/e technische universiteit eindhoven

Contributions

Conclusions/Contributions

- Distinguish control-flow, data-flow, reference-flow
- It is possible to design 'style-agnostic components‘
- Styles are a means for glueing together components



CVSscan

Visualization of Software Evolution

Lucian Voinea
Alex Telea

– Benevol 2005 –
Eindhoven, Netherlands
27.05.2005

Visit us @ www.win.tue.nl/vis/

Challenge

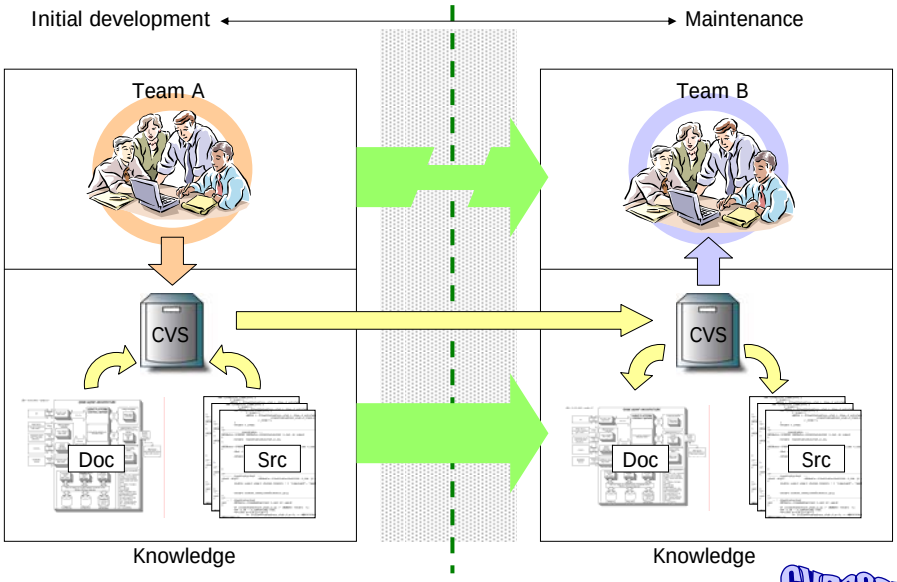
Maintenance costs / Total costs > 90%
Erlikh, L. (2000). "Leveraging legacy system dollars for E-business" (IEEE) IT Pro, May/June 2000, 17-23.

Bug discovery = 70 – 90% time
Stephen G. Eick; "CH21: Maintenance of larger systems" in "SV: Programming as a multimedia experience", MIT Press, 1998, p. 315

Code Analysis = 47 % time
Source: "Software Quality: Producing Practical, Consistent Software" Mordecai Ben-Menachem & Garry S. Marliss, Thomson Computer Press, 1997.

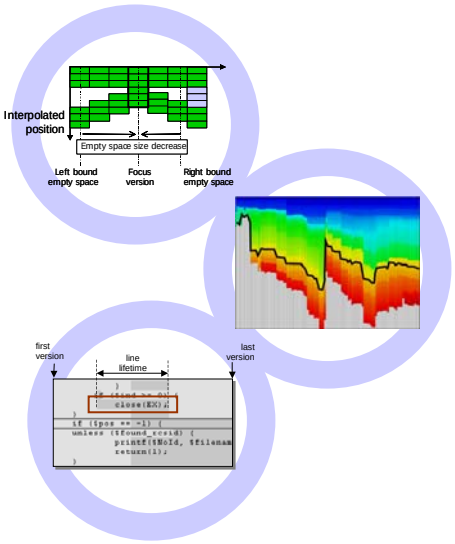
Visit us @ www.win.tue.nl/vis/

Challenge



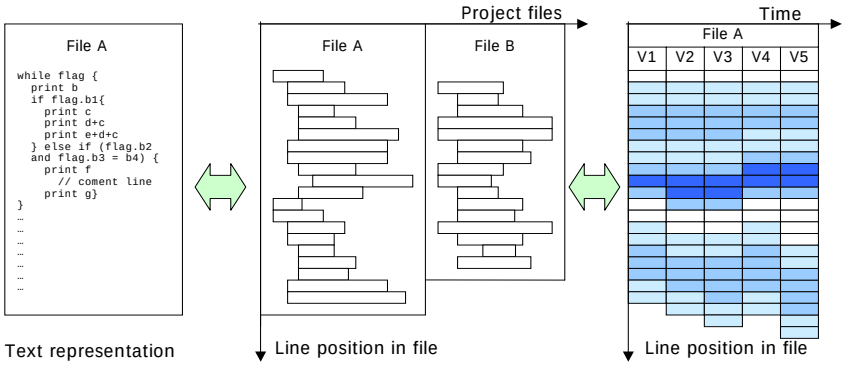
Visit us @ www.win.tue.nl/vis/

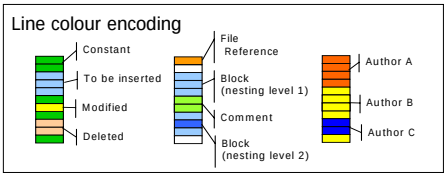
Outline



Visit us @ www.win.tue.nl/vis/

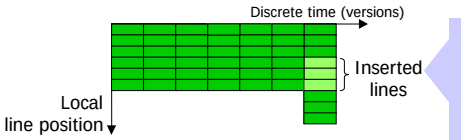
Line encoding

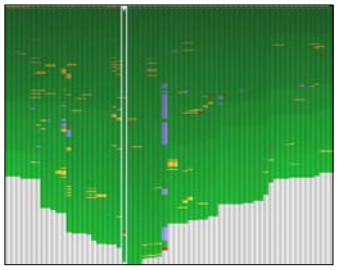




Visit us @ www.win.tue.nl/vis/

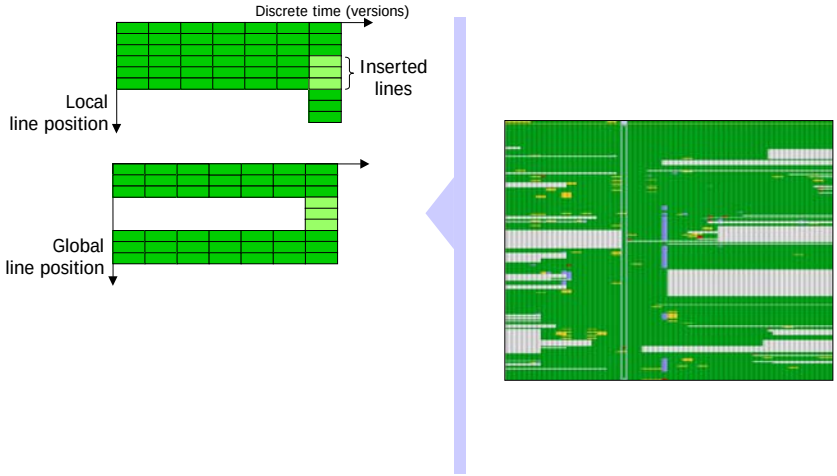
Version layout – file based





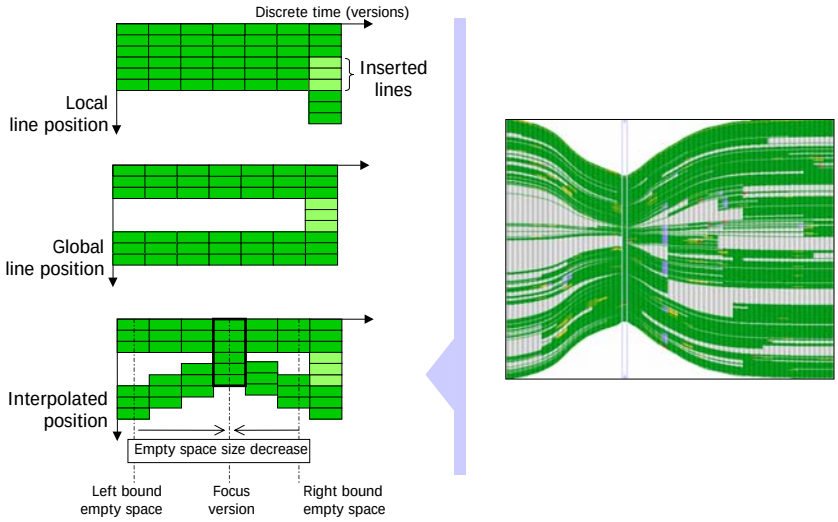
Visit us @ www.win.tue.nl/vis/

Version layout – line based



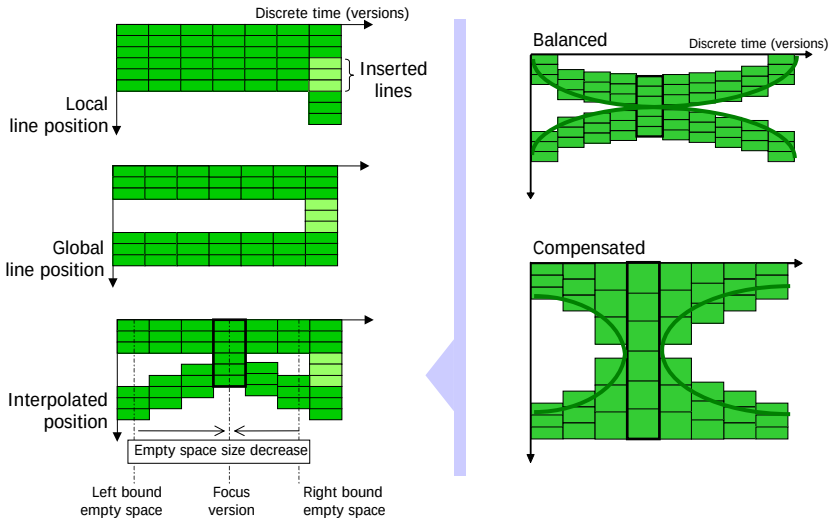
Visit us @ www.win.tue.nl/vis/

Version layout – interpolated



Visit us @ www.win.tue.nl/vis/

Version layout – interpolated

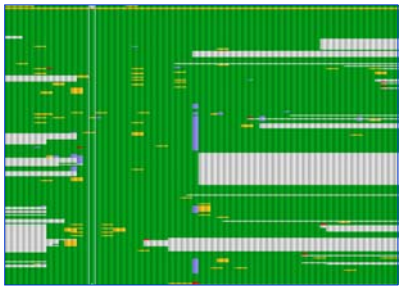


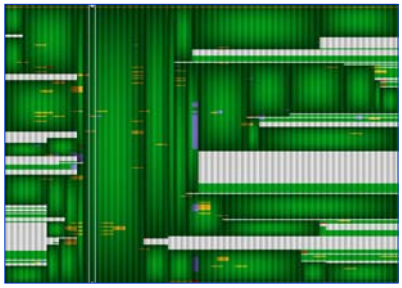
Visit us @ www.win.tue.nl/vis/

Visual improvements

<u>Antialiasing</u> : Position based (to preserve structure)

<u>Stable block detection</u> : Cushion based





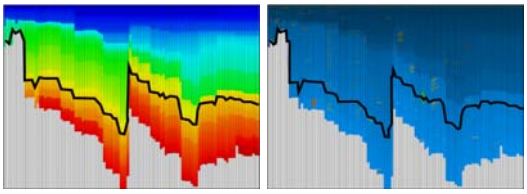
Visit us @ www.win.tue.nl/vis/

Interaction

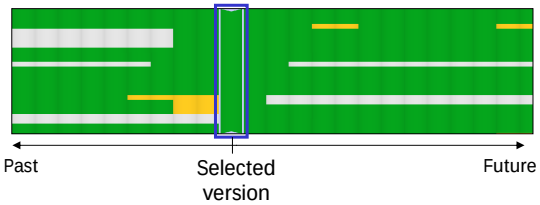
Zoom : Custom + predefined (fit to screen, fit to line)

Selection : Evolution interval selection

Navigation :



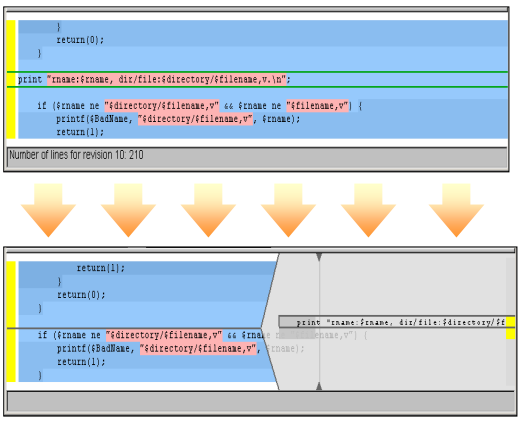
Filtering :

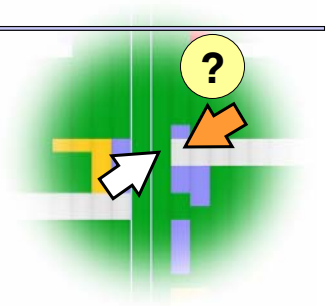


Visit us @ www.win.tue.nl/vis/

Interaction

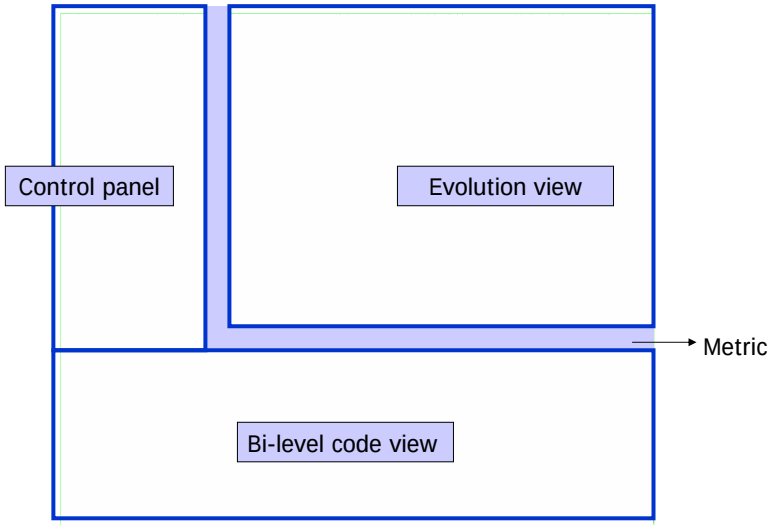
Details on demand :





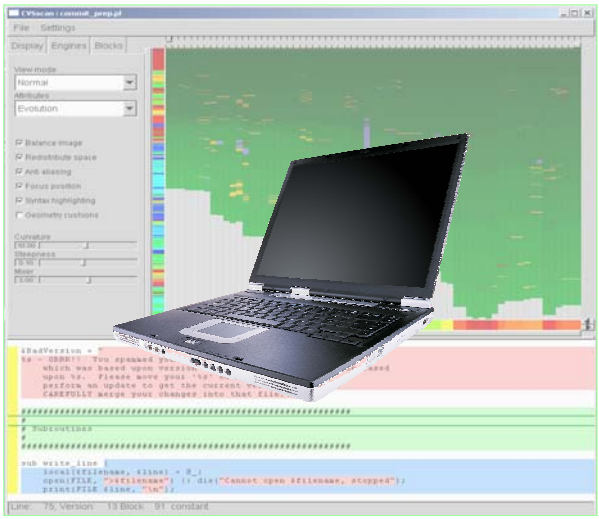
Visit us @ www.win.tue.nl/vis/

Tool presentation



Visit us @ www.win.tue.nl/vis/

Demo



Visit us @ www.win.tue.nl/vis/

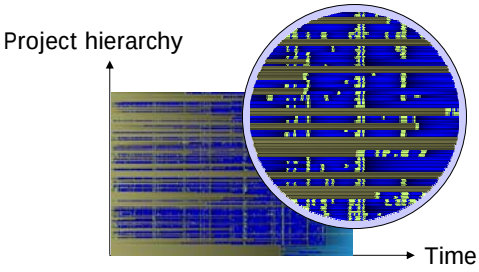
Conclusions

- Conveys process information embedded in the source code evolution records
- Offers version centric views on the system for relative assessment
- Appropriate mechanisms to navigate and browse the representation

Visit us @ www.win.tue.nl/vis/

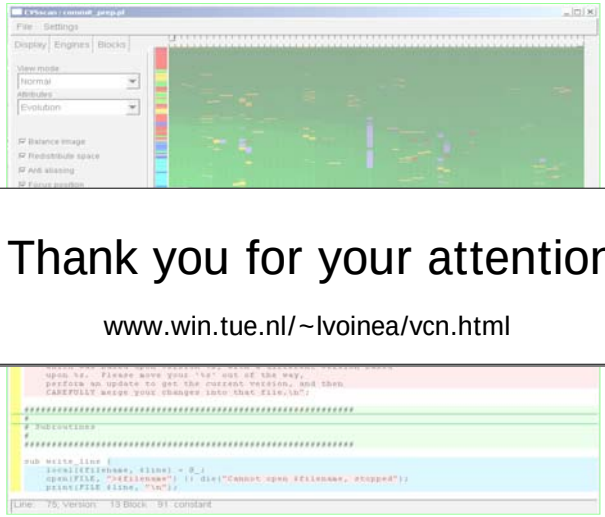
Future work

- Visualize the evolution of more files in parallel



- Enrich the information about file structure
  - requires language specialization

Visit us @ www.win.tue.nl/vis/



Thank you for your attention !

www.win.tue.nl/~lvoinea/vcn.html

Visit us @ www.win.tue.nl/vis/

Bad smells and Refactorings

Let's answer 4 major questions

Peter Ebraert & Tom Mens

Answering 4 questions

- Are there relations between bad smells?
- Are there relations between refactorings?
- Are there more relations between bad smells and refactorings?
- Do refactorings really take away bad smells?

Relations between bad smells?

- Model each bad smell as an IV
- Mine for relations
  - Find relations between bad smells
  - Check why intuitive relations do not hold
    - Fine-tune Bad smells Model

Relations between refactorings?

- Model the refactoring prerequisites as IVs
- Mine for relations on a SW application
  - Find relations between refactorings

Relations between bad smells and refactorings?

- Model each bad smell as an IV
- Model the different refac. prerequisites as IVs
- Mine for relations on a SW application
  - Answers the question whether we really can apply all refactorings (Fowler suggested) for solving a bad smell

Do refactorings really take away bad smells?

- Model each bad smell as an IV
- Find all bad smells in all CVS versions of a SW application and establish a "bad smell occurrence curve"
- Check the CVS comments to see wether refactorings were carried out on which points

Understanding Change

Where do we look at?

Filip Van Rysselberghe

Benevol 2005

Background

which change operations (and sequences) are applied?

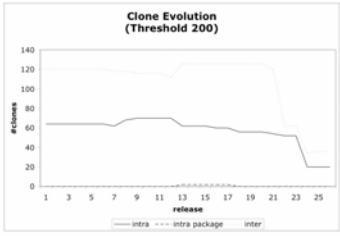
- Many changes
  - Which ones are relevant?
- Relevant changes
  - Changes to remove design problems
  - Changes that influence the evolution
  - Refactorings

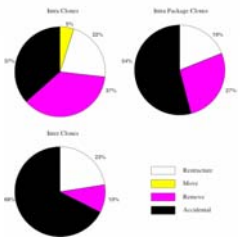
Approaches

- Problem-up
  - Focus a change when it removes a problem
- Evolution-up
  - Focus a change when its evolution changes
- Rationale-up
  - Focus a change when the developer tags it

Problem-up

	R1			R2			R3			R4			R5			R6		
CookieTools.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
Ajp12ConnectionHandler.java	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0
IncludeGenerator.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
EmbeddedServletOptions.java	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0
FileRealmTool.java	4	0	0	4	0	0	4	0	0	0	0	0	0	0	0	0	0	0
SimpleTcpEndpoint.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
DefaultServlet.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
WarFileServlet.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
ForwardGenerator.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
Parser.java	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0
PoolTcpEndpoint.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
CookieUtils.java	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
TagLibraryInfoImpl.java	4	0	0	4	0	0	4	0	0	4	0	0	4	0	0	4	0	0
WebDescriptorFactoryImpl.java	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0
LocaleToCharsetMap.java	2	0	0	2	0	0	2	0	0	2	45	0	2	45	0	2	45	0
ApacheConfig.java	0	0	0	0	0	0	0	0	0	2	0	0	2	0	0	2	0	0
MimeMap.java	0	0	0	0	0	0	0	0	0	2	45	0	2	45	0	2	45	0
ContextManager.java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
DefaultCMSetter.java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LocaleToCharsetMap.java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MimeMap.java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ApacheConfig.java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JservConfig.java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ErrorHandler.java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	18	8	0	18	8	0	18	8	0	18	50	0	18	90	0	18	92	2



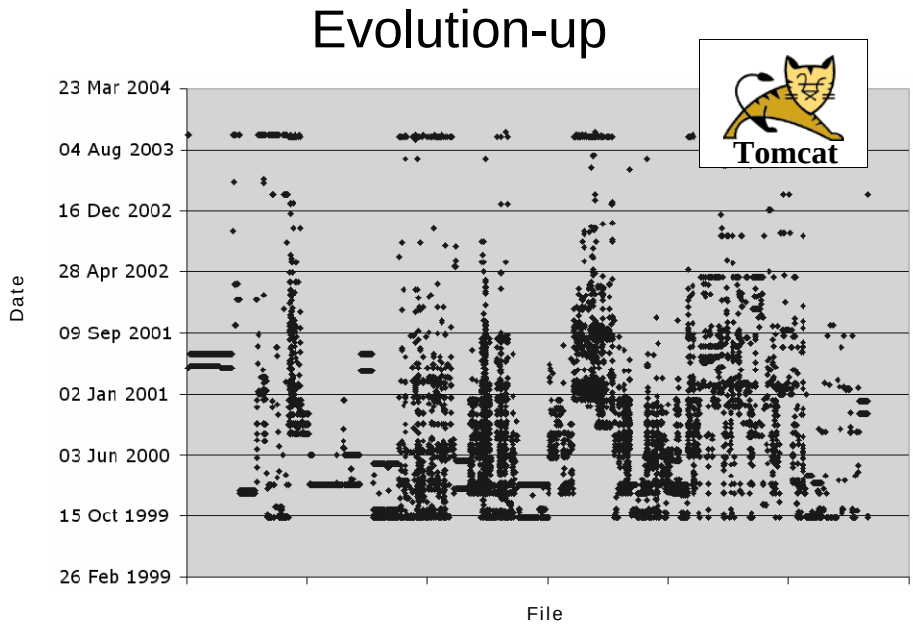


Experimented with other code smells as well.

Problem-up experiences

- Manual validation
  - Time intensive
  - Need for a high level representation
  - →Can graphs help us out?
- Change rationale
  - Why is not always as obvious
  - Link with additional info! Developers!

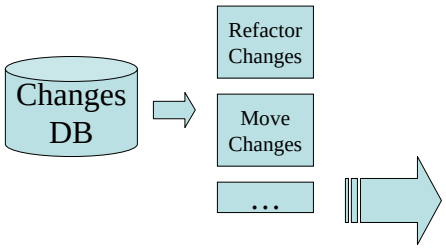
Evolution-up



Evolution-up experiences

- Can we put it in a number?
  - Change rate?
  - Patterns?
- Suitable to study the effect of …
  - A change?
  - A code smell/problem?
- Links to the rationale
  - Learns us more about the why!
- Still have to study actual changes
  - High-level change representation!

Rationale-up



Quality	Amount
1. Easier	1. Move
2. Move	2. Improve
3. Refactor	3. Refactor
	4. Clean

Tomcat:
- Refactor dependencies
- Move & Update

project	MRs	refactorMRs	percentage
ArgoUML	8127	210	5%
Gaim	9906	4	0,04%
Jakarta-Tomcat	4098	60	1,40%
Jboss	7437	63	0,84%
PostgreSQL	20013	21	0,10%

Rationale-up Experiences

- Differences in CVSquality
  - Quick assessment of messages
- Branching is a problem
- High level change view!
- Rationale!

Conclusion